Szabad Szoftver Konferencia és Kiállítás 2012

követő kiadvány

fsf.hu alapítvány

ISBN 978-963-89486-1-8

9 789638 948618

A KONFERENCIA TÁMOGATÓI

Kiemelt támogatók

Kiemelt médiatámogató

Óbudai Egyetem

fsf.hu alapítvány

hwsw
www.hwsw.hu

Támogatók

BalaBit IT Security

DOCLERWEB

ANDREWS IT ENGINEERING

MULTIP

monosX

SUSE

Médiatámogatók

HUP

ubuntu hu

NetACADEMIA
A LEGJOBBAKAT TANÍTJUK.

SZABAD SZOFTVER KONFERENCIA ÉS KIÁLLÍTÁS 2012

követő kiadvány

Budapest, 2013

A közreműködők névsora:

Mátó Péter
fő szervező – stratégia, koordináció, web
Torma Hajnalka
helyszín, catering, előadók és nonprofit kiállítók szervezése, tördelés
Bőle György
támogatók és kiállítók szervezése
Erdei Csaba
stratégia, pénzügyek
Baráth Gábor
követő kiadvány borítója
Kelemen Gábor
követő kiadvány és web korrektúrája
Németh László
követő kiadvány tördelése
Rózsár Gábor
fotózás
Szántai István
követő kiadvány korrektúrája
Tímár András
követő kiadvány korrektúrája
Torma László
sajtó, kommunikáció
Varga Csaba Sándor
technikai feltételek, ajándékok
Zahemszky Gábor
követő kiadvány és web korrektúrája
Zelena Endre
követő kiadvány tördelése

ISBN 978-963-89486-1-8

9 789638 948618

FSF.hu Alapítvány
URL: http://www.fsf.hu/
E-mail: fsf@fsf.hu

A kiadvány tördelése LibreOffice 4.0.1-ben készült, a borítóhoz GIMP-et használtunk.

TARTALOMJEGYZÉK

MENTSÜK, AMI MENTHETŐ – BEVÁLT MENTÉSI MEGOLDÁSOK LINUXRA

Bodnár Csaba
bocs@multip.hu

Kivonat

Az előadás áttekinti a Linux-alapú rendszerek különböző mentési lehetőségeit a legegyszerűbb tar mentéstől a grafikus felhasználói felülettel rendelkező komplett mentési megoldásokig. Felhívja a figyelmet a szokásos problémákra, és beszél az egyes megoldások előnyeiről és hátrányairól.

Tartalomjegyzék

1. Bevezetés

Azzal mindenki tisztában van, hogy a mentésnél nincs unalmasabb téma. |-(
Azzal is tisztában kell azonban lennünk, hogy fontosabb sincs: egy cég egész további működőképessége múlhat azon, hogy egy véletlen törlés vagy egy hardver meghibásodása esetén van-e hova nyúlni, vannak-e mentéseink, amiből rekonstruálni tudjuk a korábbi (és lehetőleg minél későbbi) előző állapotot. De ha csak egy magánszemély notebookjáról van szó: az adatvesztés ott is egy hosszabb, a gépen végzett munka eredményét vagy egy elkészült dokumentumot tehet tönkre – emiatt nem tudjuk határidőre leadni az anyagot, ami dupla munkát jelent, és amivel pénzt, reputációt, stb. veszíthetünk.
Szóval foglalkozzunk érdekes dolgokkal gépünkön, vagy kevésbé érdekesekkel, ahogy a munkánk éppen megkívánja, de egy dologról ne feledkezzünk meg: MENTSÜNK!
És lehetőleg nyílt forrású szoftverekkel … ;)

2. Mentés vagy archiválás?

Egy, az interneten talált definíció szerint:
- backup: for high-speed copy and restore to minimize the impact of failure, human errors or disaster
- file archiving: to effectivly manage data for retention and long-term access and retrieval

Ez alapján az előadás témája a mentés, nem az archiválás.

3. Adattárolási modellek (Data repository models[1])

3.1. Strukturálatlan

Egyszerűen egy külön helyen tároljuk a különböző mentéseinket, mindenféle rendszerezés nélkül.

3.2. Teljes mentés

Minden alkalommal az összes adatot mentjük.

3.3. Inkrementális mentés

Készül egy teljes mentés (heti mentési ciklus esetén pl. a hét első napján), majd az egyes mentések mindig csak az előző napihoz képest a változásokat tartalmazzák.

3.4. Differenciális mentés

Készül egy teljes mentés (heti mentési ciklus esetén pl. a hét első napján), majd az egyes mentések az azóta történt változásokat tartalmazzák.

[1] http://en.wikipedia.org/wiki/Backup#Data_repository_models

3.5. Reverse delta

Tároljuk az egyes fájlok aktuális állapotát, továbbá azokat a különbségeket (delta), amikből vissza lehet állítani a jelenlegi állapotból a korábbi állapotokat (reverse).
Tipikusan így működnek a verziókövető rendszerek.

3.6. Folyamatos adatvédelem (Continuous Data Protection, CDP)

Wikipédia szerinti leírás: Rendszeres, ütemezett mentés helyett ebben az esetben a rendszer minden változást azonnal naplóz (journaling) a host rendszerre. Ez általában a bájt- vagy blokkszintű különbségek mentését jelenti, nem pedig fájlszintű különbség-mentést. Ez annyiban különbözik a diszk tükrözéstől, hogy itt lehetséges a log visszatekerése (roll-back), és így az adatok egy régebbi állapotának visszaállítása.

4. Mentési eszközök

4.1. Mágnesszalag

Korábban SLR, DAT, DLT, ma leginkább LTO:
- korábban jobb kapacitás/ár arány, mint a diszkeknél
- hordozhatóság
- nagy adatmennyiség tárolható
- a mai szalagos egységek nagy sebességűek
- szekvenciális elérés

4.2. Lemez

- egyre olcsóbb
- változatos interfészek (SCSI, USB, SATA, ...)
- léteznek keretbe szerelt, cserélhető diszkek kifejezetten mentési célra
- ma már épülnek diszk alapú tárolóegységek, amelyek a deduplikációt is lehetővé teszik

4.3. Optikai tárolók (CD/DVD/Blu-ray)

- olcsó
- korlátozott kapacitás
- korlátozott írási sebesség
- hordozhatóság

4.4. Távoli mentési szolgáltatás (Remote backup service)

- a nagy sebességű internet korában reális alternatíva lehet
- adatbiztonság?
- illetéktelen hozzáférés lehetséges?
- feltöltéskor is, tároláskor is titkosítsunk

5. Élet a mentési megoldások előtt

5.1. Unix-alapú rendszerekben

A Unix-alapú rendszerekben a 70-es évek óta léteznek azok a segédprogramok, amelyekkel teljes, megbízható mentéseket készíthetünk:

- a cpio-val és a tarral az összecsomagolást
- a compress-szel a tömörítést
- a find segítségével az inkrementális és a differenciális mentést
- a különböző mt, tape, stb. parancsokkal a szalagkezelést (betöltés, kivétel, előretekerés, visszatekerés, szalagjelek (tape mark) kezelése: jel írása, ugrás szalagjelre, stb.)
- az mtx, stb. parancsokkal a szalagkönyvtár kezelését
- rsh-val, rcp-vel, ftp-vel pedig a távoli szerverre történő mentést lehetett megoldani.

5.2. Linuxos rendszerekben

A 90-es években a Linuxban, BSD-ben megjelenő újabb segédprogramok pedig további lehetőségeket adnak:

- **gzip, bzip2** – a korábbinál jobb minőségű tömörítés
- **tar -z, -j** opciókkal a tömörítés egyetlen paranccsal elvégezhető
- **rsync** – távoli szerverre történő fájlrendszer szinkronizálás

5.3. Problémák ezekkel a mentésekkel

- **Inkonzisztencia:** már egy egyszerű, statikus webtartalom esetén előfordulhat (ha valaki éppen mentés közben frissíti), hogy pl. a html-oldal mentésre kerül, de az abban hivatkozott kép már nem – visszatöltéskor probléma
- **Adatbázis-inkonzisztencia:** az adatbázishoz tartozó egyes fájlok különböző időpontban kerülnek mentésre – teljes inkonzisztenciát, használhatatlanságot okozhat!
- **Grafikus frontend hiánya:** sok szerver mentése esetén a mentések áttekintését, ütemezését, a lefutások sikerességének ellenőrzését, ad hoc mentések készítését leegyszerűsíti.
- **Nem segít az „új kor új igényeinek” megoldásában:** pl. virtuális gépek image-szintű mentése

6. Megoldások

6.1. Lehetséges megoldások az adatbázis inkonzisztencia problémára

- Mentés előtt adatbázis dump készítése – azt mentjük. Egy bizonyos méret felett nem kivitelezhető.
- Mentés idejére adatbázis leállítása: 7×24 órás üzem esetén nem lehetséges.
- Adatbázis másik üzemmódba rakása a mentés idejére. Ez alatt az idő alatt a változások redo logokban gyűlnek.
- Partíció- vagy fájlrendszer-szintű pillanatfelvétel (snapshot) készítése. Ezt mentjük.
- Mentési szoftver használata a megfelelő adatbázis-pluginnal.

6.2. Gyártófüggő fizetős megoldások

- EMC2 (korábban Legato) NetWorker – már a kisvállalati kezdő csomag is nagyon drága
- Tivoli – összeintegrálva a tárolórendszer (storage) kezelés és a mentése
- ArcServe
- NetVault
- Arkeia

6.3. Nyílt forráskódú megoldások

A nyílt forrású mentési szoftverek közül a legnépszerűbb a Bacula és az Amanda. Mindkettőnek van ingyenes és fizetős verziója.

Bacula

A Bacula nagyon (talán túlságosan is) jól particionált. Az egyes funkcionalitások élesen el vannak választva egymástól. Így viszont szükségtelenül bonyolult, „pilótavizsgás”, egy egyszerűbb mentés beüzemelése is sokáig tart vele. „Bacula is quite a beast” írja valaki emiatt az interneten. Továbbá sokak szerint kiforratlan még.

1. ábra: Bacula application interactions

Többféle, többé-kevésbé jól használható grafikus felülettel rendelkezik: ezek közül egyik a qt-alapú BAT (Bacula Admin Tool), másik a Bweb webes interfész – ez azonban csak a fizetős verzióban van benne. :(
A Bacula (bacula.org) mögött a Bacula Systems nevű svájci cég van, a fizetős verzió náluk érhető el. Sajnos bizonyos funkciók csak a fizetős verzióban érhetőek el.
Nincs benne az ingyenes verzióban:

- Delta Plugin
- SAN Shared Storage Plugin
- Client-free VMware Plugin
- Microsoft Exchange VSS Plugin
- Microsoft SQL Server VSS Plugin
- NDMP Plugin

Amanda

Mentés egy központi helyen, ugyanarra a szerverre vannak kapcsolva a mentési eszközök is. Egyszerű a felépítése: szerver szoftver, pluginek és agentek.
Az Amanda (a Baculától eltérően) nem saját mentési formátumot használ, hanem olyan standard segédprogramokra épül, mint a tar és a gzip.
Nem találtam hozzá grafikus menedzsment felületet: A web alapú felhasználói interfész csak a fizetős csomagban van benne, amelyet a Zmanda Inc. fejleszt. A fizetős és az ingyenes verzió közötti különbségek:

6.4. Licencelés

Legyen szó akár a Baculáról, akár az Amandáról, sajnos a jó minőségű grafikus felhasználói felület, illetve a különlegesebb mentéseket (adatbázisok, virtuális gépek, storage, stb.) lehetővé tevő pluginek csak a fizetős verzióban találhatóak meg. Ráadásul a szoftverek csak az előfizetési időszak alatt használhatóak. Tehát hiába vesszük meg őket, ha lejár az időszak, azzal nem csak a technikai támogatás, frissítések, stb. lehetősége szűnik meg, hanem maga a szoftver (a szerver oldali rész, a pluginek, stb.) sem használható tovább.
Úgyhogy (és ez akár összefoglalója is lehet az előadásnak) az mondhatjuk, hogy léteznek jól használható nyílt forráskódú mentési megoldások, de azok kényelmi funkciói csak a fizetős verziókban találhatók meg. :(

WEBES ÉS MOBIL ALKALMAZÁSOK F#-BAN WEBSHARPERREL

Gránicz Ádám
IntelliFactory

Kivonat

A web alapú alkalmazások fejlesztése rendkívüli iramban fejlődik: a JavaScript, mint a web nyelve manapság már felhasználható nemcsak webalkalmazások írására, hanem web alapú mobil, sőt desktop operációs rendszereken futó alkalmazások írására is.

A WebSharper egy az F# nyelvhez fejlesztett, nyílt forráskódú keretrendszer, amely webes és mobil web alkalmazások készítését teszi lehetővé egyetlen F# kódbázisból.

Tartalomjegyzék

1. Bevezető

A web alapú alkalmazások fejlesztése rendkívüli iramban fejlődik: a JavaScript, mint a web nyelve manapság már felhasználható nemcsak webalkalmazások írására, hanem web alapú mobil, sőt desktop operációs rendszereken futó alkalmazások írására is. Emellett a kiszolgáló-oldali felhasználása is jelentős (lásd NodeJS), így szinte körbeérve a kliens-szerver alkalmazások fejlesztéséhez szükséges és használt nyelvek és technológiák arzenálját.

A web alapú fejlesztések fő előnye az újrahasználhatóság: ugyanaz a kódbázis akár csekély módosítással áthelyezhető más platformokra, pl. egy HTML5-ben megírt webalkalmazásban elegendő csupán a felhasználótól jövő interakciók kezelését megváltoztatni (pl. a mobil gesztúrák kezelésével kiterjeszteni), a kód többi része kisebb-nagyobb változtatással megállja a helyét az új platformra (mobil vagy desktop) készített alkalmazás részeként.

2. A WebSharper keretrendszerről

A WebSharper (http://websharper.com) egy az F# nyelvhez fejlesztett, nyílt forráskódú keretrendszer, amely webes és mobil web alkalmazások készítését teszi lehetővé egyetlen F# kódbázisból. A rendszer alapját képező F#→JavaScript fordító feladata az F# kódból a kliens oldali részek (amelyek speciális attribútummal vannak annotálva a kódban) átfordítása JavaScriptre, a hozzájuk tartozó esetleges kliens oldali függőségek (CSS, JS file-ok) automatikus felderítése, és az ezekhez tartozó egyéb metainformációk elérhetővé tétele, ezzel elősegítve újrahasználható komponensek, ill. az azokból felépített webalkalmazások fejlesztését. További nagy előnye, hogy a kliens-szerver hívások (amelyek egyszerű függvényhívások két, kliens és szerver attribútumokkal ellátott függvény között) is automatikusan fordításra kerülnek, így a fejlesztőknek nem okoz többé fejtörést, hogy miként adhatók át értékek a két tier (a kliens és a szerver) között.

A keretrendszer másik szerves részét az F# nyelvbe ültetett webes absztrakciók képezik. Ezen absztrakciók nagyban hozzájárulnak ahhoz, hogy az egyes és tipikus webes feladatkörök (pl. validálható űrlapok fejlesztése a felhasználói adatok gyűjtésére) tömör, absztrakt, és egymásba ágyazható módon legyenek teljesíthetők, mindamellett a típusrendszerben történjenek, így biztosítva, hogy robusztusabb és szemantikailag értelmes alkalmazások készüljenek, minimális hibalehetőséggel, amelyet esetlegesen futás közben kellene teszteléssel kiszűrni.

A keretrendszer fent említett alkotórészei mellett a kibővíthetőség két fronton jelenik meg: az úgynevezett „proxy”-k szintjén, amelyek .NET könyvtárak JavaScriptre történő fordításában segítenek, illetve a „stub”-ok szintjén, amelyek az ellenkező irányban működnek, és JavaScript könyvtárak .NET-ben elérhető API-jait hivatottak biztosítani. Maga a keretrendszer alapból nagyszámú proxy-val van ellátva, amelyekkel szinte a teljes F# sztenderd könyvtárhalmaz és jó néhány .NET sztenderd könyvtár használata lehetséges, ezek JavaScriptre fordítása minden probléma nélkül automatikusan biztosított. Ennek fordítottja is hasonló helyzetet mutat: alapból számos „kötelező” JavaScriptes könyvtár (pl. Ecmascript, HTML5, jQuery, stb.) érhető el közvetlenül F#-ból.

3. WebSharper Mobile

A WebSharper Mobile a WebSharper keretrendszer azon rétege, amely a mobil webalkalmazások készítését segíti elő. Ezen rétegben két fő extra funkcionalitás érhető el az alapképességeken kívül: egyrészt különböző mobil képességeket és funkcionalitásokat (gyorsulás adat, GPS pozíció, kamera, stb.) tesz JavaScriptből elérhetővé, másrészt pedig a már lefordított JavaScript alkalmazásokat csomagolja be különböző mobil platformokon futó natív alkalmazásokká.

4. WebSharper template-ek

Egy WebSharperrel készült webalkalmazás bármilyen tartalmat szolgáltathat egy-egy adott URL-en keresztül. Webes tartalom kiszolgálására két fő mechanizmust kínál: statikus és dinamikus template-eket.

A statikus template-ek lényege, hogy maga a webes tartalom (amely XML formátumban kerül egy-egy projektbe) automatikusan F# kóddá kerül átfordításra, amely az alkalmazás forrásfájljával együtt kerül fordításra. Így a statikus template-ekben lévő esetleges hibák fordításidőben jelentkeznek. Ugyanakkor ez némi hátrányt is jelent: hiszen nem lehet még apróbb változtatásokat sem elvégezni a template-eken az alkalmazás teljes újrafordítása nélkül.

A dinamikus template-ek az előbb említett problémához úgy próbálnak megoldást adni, hogy a template tartalmát dinamikusan, futásidőben töltik be, és végzik el rajta a szükséges tartalom helyettesítését. Így bárminemű változtatás megvalósítható a template-ben újrafordítás nélkül, amelyek azonnal láthatók az alkalmazásban, ugyanakkor az esetleges rejtőző hibák csak futásidőben (az első oldal betöltésénél) kerülnek a felszínre.

5. WebSharper siteletek

A WebSharper siteletek webalkalmazásokat modelleznek, erősen tipizált, egymásba ágyazható F# értékként. A siteletek különböző kombinátorok segítségével hozhatók létre, növelhetők funkcionalitással, és ágyazhatók egybe. Egy sitelet egy adott Action típus felett van definiálva, amely leírja, hogy milyen „absztrakt” belépési pontoknak felel meg a végső sitelet-alapú alkalmazás.

Az alábbi Action típus a sztenderd WebSharper installáció sitelet-alapú példa webalkalmazás projektjében van definiálva:

```
type Action =
    | Home
    | Contact
    | Protected
    | Login of option<Action>
    | Logout
    | Echo of string
```

Itt jól látható, hogy hat különböző „funkcionalitást” definiálunk egy diszkriminált unió típusban. Erre a típusra készíthetünk különböző tartalmakat. Az alábbi példa egy dinamikus template-ből képzett, egy példányosítható „lyukból” álló függvény, a WithTemplate, segítségével hoz létre a Home oldalhoz tartozó HTML tartalmat:

```
/// The pages of this website.
module Pages =
    /// A helper function to create a hyperlink
    let ( => ) title href = A [HRef href] -< [Text title]
```

```
    /// The home page.
    let HomePage : Content<Action> =
        WithTemplate "Home" <| fun ctx ->
            [
                H1 [Text "Welcome to our site!"]
                "Contact" => ctx.Link Action.Contact
            ]
    ...
```

Ezt aztán kiajánlhatjuk, mint egy singleton sitelet a következő kóddal:

```
let home =
    Sitelet.Content "/" Action.Home Pages.HomePage
```

Itt a Sitelet.Content függvény segítségével létrehozunk egy siteletet, amely a gyökér URL-re hallgat, és az odaérkező kérésekre a fent definiált Pages.HomePage tartalmat szervírozza.

A fenti módszer tökéletes arra, hogy alacsony szintű kontrollt gyakoroljunk a létrehozott sitelet által kezelt URL térre, azonban ez a legtöbb esetben nem szükséges: hiszen a legfőbb célunk az hogy ne legyen rossz URL az alkalmazásunkban, nem pedig az, hogy mik legyenek az adott URL-ek. Ezt mutatja be a következő kódrészlet, amely a fent definiált Action típus többi alakzatához is rendel tartalmat:

```
// An automatically inferred sitelet for the basic parts.
let basic =
    Sitelet.Infer <| fun action ->
        match action with
        | Action.Contact -> Pages.ContactPage
        | Action.Echo param -> Pages.EchoPage param
        | Action.Login action -> Pages.LoginPage action
        | Action.Logout ->
            // Logout user and redirect to home
            UserSession.Logout ()
            Content.Redirect Action.Home
        | Action.Home -> Content.Redirect Action.Home
        | Action.Protected -> Content.ServerError
```

Itt érdemes megfigyelni, hogy az Action.Logout alakzat nem kap külön tartalmat, hanem kilépteti az éppen aktuálisan bejelentkezett felhasználót és átirányít a Home oldalra. Ez az átirányítás nem string URL-eken keresztül történik, hanem az Action típus megfelelő alakzatán keresztül, ezzel biztosítva a típusbiztonságot.

Szintén megfigyelendő az Action.Protected alakzat kezelése: egyszerűen szerver hibával elutasítjuk ennek szervírozását. Ezt azért tesszük, mert mint „védett” tartalmat is tudjuk definiálni a következőképpen:

```
// A sitelet for the protected content.
let authenticated =
    let filter : Sitelet.Filter<Action> =
        {
            VerifyUser = fun _ -> true
            LoginRedirect = Some >> Action.Login
        }
    Sitelet.Protect filter
    <| Sitelet.Content
         "/protected"
         Action.Protected
         Pages.ProtectedPage
```

Itt egy filtert definiálunk, amellyel a sztenderd beléptetésen kívül elutasíthatunk bizonyos felhasználókat, ill. meghatározhatjuk, hogy melyik „oldalra” ugorjunk vissza a beléptetéshez, amely szükséges az adott tartalom megjelenítéséhez. Esetünkben nem támasztunk semmilyen extra követelményt, és alapból a Login oldalra irányítjuk a felhasználót a beléptetéshez.

Ennek a filternek a segítségével már könnyedén „védhetjük” az adott védett tartal-

munkat, és képezhetünk belőle egy siteletet a Sitelet.Protect függvény segítségével. Az így definiált siteleteket pedig egyszerűen egymásba ágyazhatjuk a Sitelet.Sum alkalmazásával:

```
let EntireSite =
    // Compose the above sitelets into a larger one.
    Sitelet.Sum
        [
            home
            authenticated
            basic
        ]
```

Az így létrejövő sitelet már kiajánlható, mint „webalkalmazás” és egyszerűen szervírozható bármely ASP.NET-kompatibilis konténerben.

6. WebSharper formletek

A siteletekhez hasonlóan, a WebSharper keretrendszer lehetőséget ad webes űrlapok F# értékként való ábrázolására is. Az így létrejövő formletek szintén tetszőlegesen egymásba ágyazhatók, és erősen tipizálva jelennek meg a típusrendszerben.

A sztenderd formletek mellett két fő formlet halmazt kapunk a keretrendszer részeként: dependens formleteket, amelyek képesek deklaratívan, programmatikusan kifejezni egyes kontrollok közötti függőségeket, ill. a flowletek, amelyek varázsló-stílusú, egymás után következő formleteket képesek együtt kezelni, mint egyetlen F# érték.

A formletek egymásba ágyazásának absztrakt módja a következő pszeudokóddal írható le:

```
Formlet.Yield (fun v_1 v_2 ... v_n -> <compose all v_i's>)
<*> formlet_1
<*> formlet_2
...
<*> formlet_n
```

Egy-egy formlet alapja mindig egy adott kontroll, pl. egy egyszerű szöveges input box. Ennek testreszabásához rengeteg függvény áll rendelkezésre. Például a következő formlet egy címkével ellátott szöveges kontrollhoz egy nem-üres kliens-oldali validációt rendel, amelyet egy ikon formájában ad a felhasználónak tudtára:

```
let TB label msg =
    Controls.Input ""
    |> Validator.IsNotEmpty msg
    |> Enhance.WithValidationIcon
    |> Enhance.WithTextLabel label
```

Ennek a formlet függvénynek a felhasználásával már könnyedén készíthetünk komplexebb űrlapokat:

```
type Person = { Name: string; Email: string }
[<JavaScript>]
let PersonFormlet () : Formlet<Person> =
    let nameF = TB "Name" "Empty name not allowed"
    let emailF = TB "Email" "Please enter a valid email address"
    Formlet.Yield (fun name email -> { Name = name; Email = email })
    <*> nameF
    <*> emailF
    |> Enhance.WithSubmitAndResetButtons
    |> Enhance.WithLegend "Add a New Person"
    |> Enhance.WithFormContainer
```

Ez a formlet már az általunk definiált Person típusú értékeket képes visszaadni, és a következőképpen néz ki az alap renderelő által:

7. WebSharper bővítések

A WebSharper keretrendszer opcionális „bővítéseket”, azaz különböző JavaScript könyvtárak .NET reprezentációit is képes kezelni, ezzel lehetővé téve, hogy a fejlesztők azokat F#-ból tudják megcímezni, mindenféle JavaScript kódolás helyett.

Az előző fejezetben bemutatott formlet esetén például számos opcionális bővítmény áll rendelkezésre, jQuery UI, Yahoo UI, Ext JS, jQuery Mobile, stb. könyvtárak kontrolljait felhasználva a formletek renderelésére.

Ezek mellett számos térképmotor, pl. Google Maps vagy Bing Maps, vizualizációs könyvtárak, mint pl. Infovis, Protovis, Google Visualization, vagy HTML5-ös könyvtárak pl. WebGL, O3D, és GlMatrix is rendelkezésre állnak, melyek segítségével impresszív alkalmazások készíthetők.

8. Mobil alkalmazások

Az előzőkben tárgyalt absztrakciók és különböző WebSharper bővítmények felhasználásával könnyedén készíthetünk komolyabb alkalmazásokat aránylag kicsi kódbázisból. Az alábbi példa egy mobil login felhasználói felületet implementál, amely második lépéseként üdvözli a frissen beléptetett felhasználót:

```
let loginSequenceF =
    Formlet.Do {
        let! username, password, remember =
            Formlet.Yield (fun user pass remember -> user, pass, remember)
            <*> (Controls.TextField "" Theme.C "Username: "
                |> Validator.IsNotEmpty "Username cannot be empty!")
            <*> (Controls.Password "" Theme.C "Password: "
                |> Validator.IsRegexMatch "^[1-4]{4,}[0-9]$" "The password
is wrong!")
            <*> Controls.Checkbox true Theme.C "Keep me logged in "
                |> Enhance.WithSubmitButton "Log in" Theme.C
        let rememberText =
            if remember then "" else "not "
        do! Formlet.OfElement (fun _ ->
            Div [
                H3 [Text ("Welcome " + username + "!")]
                P [Text ("We will " + rememberText + "keep you logged in.")]
            ])
    }
    |> Formlet.Flowlet
```

Ezt a kétlépéses flowletet könnyedén beágyazhatjuk egy jQuery Mobile-s felhasználói felületbe, a már korábban látott HTML kombinátorok segítségével:

```
Div [HTML5.Attr.Data "role" "page"] -< [
    Div [HTML5.Attr.Data "role" "header"] -< [
        H1 [Text "WebSharper Formlets for jQuery Mobile"]>
    ]
    Div [HTML5.Attr.Data "role" "content"] -< [

        loginSequenceF
```

```
 ]
    Div [HTML5.Attr.Data "role" "footer"] -< [
        P [Attr.Style "text-align: center;"] -< [Text "IntelliFactory"]
    ]
]
```

Az eredmény a következő:

Egy hasonlóan elegáns példa a Bing Maps szolgáltatását ötvözi, az alábbi kód a teljes implementációt tartalmazza:

```
type CurrentLocationControl() =
    inherit Web.Control()
    [<JavaScript>]
    override this.Body =
      let screenWidth = JQuery.Of("body").Width()
      let MapOptions =
          Bing.MapViewOptions(
            Credentials = bingMapsKey,
            Width = screenWidth - 10,
            Height = screenWidth - 10,
            Zoom = 16)
      let label = H2 []
      let setMap (map : Bing.Map) =
        let updateLocation() =
          // Gets the current location
          let loc = Mobile.GetLocation()
          // Sets the label to the current location
          Rest.RequestLocationByPoint(<<your-bingmaps-key>>,
            loc.Lat, loc.Long, ["Address"],
            fun result ->
                let locInfo =
                    result.ResourceSets.[0].Resources.[0]
                label.Text <-
                    "You are currently at " +
                         JavaScript.Get "name" locInfo)
          // Adds a pushpin at the current location
          let loc = Bing.Location(loc.Lat, loc.Long)
          let pin = Bing.Pushpin loc
          map.Entities.Clear()
          map.Entities.Push pin
          map.SetView(Bing.ViewOptions(Center = loc))
        // Keep updating your location regularly
        JavaScript.SetInterval updateLocation 1000
        |> ignore
      let map =
```

```
        Div []
        |>! OnAfterRender (fun this ->
          // Renders a map control
          let map = Bing.Map(this.Body, MapOptions)
          map.SetMapType(Bing.MapTypeId.Road)
          setMap map)

      // Returns markup for this control
      Div [
          label
          Br []
          map
      ] :> _
```

Az eredmény:

9. Összegzés

A funkcionális programozás adta absztrakciók jelentős hatékonyságnövekedést eredményeznek, így a WebSharper keretrendszer használata gyorsabb fejlesztést és kisebb kódbázist eredményez. Ezek direkt módon befolyásolják az így megírt alkalmazások fenntarthatóságát jelentős pénzt, időt és erőforrást megtakarítva. Az F# + WebSharper kombináció egy folyamatosan bővülő piac (mobil és desktop web alkalmazások) lehetőségét nyújtja, és gyors út a többplatformos megoldásokhoz, amelyet az egyre növekvő felhasználói bázis és a nyílt forráskód adta érdeklődés mértéke folyamatosan bizonyít.

A VALA PROGRAMOZÁSI NYELV

Meskó Balázs
meskobalazs@gmail.com

Kivonat

A Vala programozási nyelv egy viszonylag új fejlesztés, mely egy konkrét probléma megoldására született. Célja a modern programozási technikák biztosítása GNOME alkalmazások fejlesztése során.

A Vala ötlete Jürg Billetter svájci egyetemista fejéből pattant ki 2006-ban, amikor GObject alapú függvénykönyvtárakat használt munkájához. Ötletét megosztotta tanulótársával Raffaele Sandrinivel, akivel elkezdték a fejlesztést. A fordítóprogram működésének alapját a Vala API fájlok adják, amelyek egyszerű leírások, amelyek segítik a kódleképzést. A leginkább a többi GObject-alapú függvénykönyvtárral működik jól ez a megoldás, hiszen az ilyen könyvtárak támogatják az OOP paradigmát és a többi fejlett funkciót (pl. szignálok). Mára már szinte minden GObject függvénykönyvtárhoz van vapi leírás.

A Vala tehát egy újszerű megközelítést hozott, és ezzel felkavarta az állóvizet. Natív kódot készít, C közeli sebességgel fut, és ráadásul kevés függősége van: kötelezően mindössze a GObject és GLib függvénykönyvtárakra van szükség – ezek több platformra elérhetőek.

1. A Vala programozási nyelv

A Vala programozási nyelv egy viszonylag új fejlesztés, amely sok nyelvvel ellentétben nem a személyes kísérletezgetés, hanem egy konkrét probléma megoldására született. A nyelv küldetése pár szóban összefoglalva: *modern programozási technikák biztosítása GNOME alkalmazások fejlesztése során.* A Vala ötlete Jürg Billetter svájci egyetemista fejéből pattant ki 2006-ban, amikor GObject alapú függvénykönyvtárakat használt munkájához. Nagyon zavarta, hogy sok olyan dolgot meg kellett írnia a C programjaimban, amelyet más nyelvek automatizálnak. Ekkor találta ki, hogy szükség volna egy olyan nyelvre, amely magasabb absztrakciós szintet biztosít ezekhez a könyvtárakhoz. Ötletét megosztotta tanulótársával Raffaele Sandrinivel, akivel elkezdték a fejlesztést.

A Vala elsősorban a C# mintájára készült, de nem annak a másolata. Mindazon programozók, akik jártasak a C# vagy Java nyelvekben ismerősnek találhatják a nyelvet. A Vala viszont nem menedzselt nyelv, a forrásfájlokból C nyelvi kód generálódik a *valac* fordítóprogram segítségével. Mindezt viszonylag kevés függőséggel éri el, kötelezően mindössze a GObject és GLib függvénykönyvtárakra van szükség – ezek több platformra elérhetőek. Mivel ezek az alacsony szintű függvénykönyvtárak a GNOME platform sarokkövei, ezért a Vala nyelv ideális GNOME alkalmazások készítésére. A

fordítóprogram működésének alapját a Vala API fájlok adják, amelyek egyszerű leírások, amelyek segítik a kódleképzést. A leginkább a többi GObject-alapú függvénykönyvtárral működik jól ez a megoldás, hiszen az ilyen könyvtárak támogatják az OOP paradigmát és a többi fejlett funkciót (pl. szignálok). Mára már szinte minden GObject függvénykönyvtárhoz van *vapi* leírás.

Most lássuk, hogy eddig hogyan nézett ki a fejlesztés a GNOME platformra. Használhattuk a natív C API-t, amely eléggé bőbeszédű. Emiatt megjelentek illesztőfelületetek más programozási nyelvekhez, például a C++-hoz, Pythonhoz és Mono/C#-hoz. A C++ felület elég népszerű, de sokan kritizálják, mivel kevésbé elegáns, mint a Qt megoldása, és a dokumentáció is a Qt javára billenti a mérleget. A Python nyelv használata is nagyon elterjedt, ez talán a legjobb megoldás a három közül, de a kész program sebességgondokkal küszködhet. A Mono-t csak elvétve láthattuk GNOME alkalmazások alapjául, köszönhetően a virtuális gép okozta sebességcsökkenésnek, a függőségeinek, és nem utolsóként a vélt és valós jogi problémák miatt (ld. Tomboy → GNote).

A Vala tehát egy újszerű megközelítést hozott, és ezzel felkavarta az állóvizet. Natív kódot készít, C közeli sebességgel fut, és ráadásul kevés függősége van. Minden szépnek és jónak tűnik, de azért vannak hátrányok is persze. Mivel a végeredmény C forráskód lesz, így a Vala csak pontosan arra képes, amire egy C nyelvű program, ezt sokszor észben kell tartanunk fejlesztés közben. A szemétgyűjtő mechanizmust nem támogatja a nyelv, a C#-pal és a Javával ellentétben. Ehelyett referenciaszámlálást alkalmaz a nyelv, amely egy sokkal egyszerűbb megoldás, cserébe gyorsabb is. Az alacsony szintű lehetőségek pedig néha nem megkerülhetőek, tehát ezekhez is érteni kell. Leginkább úgy jellemezném az egész nyelvet, mint egy segítőtársat, aki megírja helyettünk az unalmas részeket, így a programozó a lényeggel foglalkozhat.

Mivel ilyen terjedelemben nehéz bemutatni egy új nyelvet, így összegyűjtöttem néhány linket az érdeklődőknek. A GNOME fejlesztői oldalon[2] megtalálható nagyon sok minden, de főleg a C API dokumentáció. A Vala gyorstalpalója[3] bemutatja a nyelv funkcióit, sok példakóddal. Ennek az online gyorstalpalónak elkészítettem a magyar fordítását, amely a blogomról letölthető (http://meskobalazs.progmat.hu/?p=1). Ha nekivágunk, akkor pedig szinte kötelező útitársunk a Vala API dokumentáció.[4] Utolsó megjegyzésként: a gyorstalpaló nincs kész, de már használható állapotban van. Elírások és tárgyi tévedések biztosan vannak, így ha valaki ilyet talál, kérem e-mailben jelezze azt nekem: meskobalazs@gmail.com. Jó kódolást!

[2] http://developer.gnome.org

[3] http://live.gnome.org/Vala

[4] http://www.valadoc.org

OPENFLOW: ÚT A SZABAD SZOFTVERES HÁLÓZATOK FELÉ?

Németh Felicián[*], Sonkoly Balázs
[*]MTA-BME Informatikai rendszerek kutatócsoport
Távközlési és Médiainformatikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem
{nemethf, sonkoly}@tmit.bme.hu

Kivonat

A szabad szoftvereknek a számítástechnika sok területén sikerült teret nyerniük, azonban a nyílt hardverek még csak szűk körben érhetők el. Az OpenFlow protokoll ebbe a körbe vonja be a hálózati útvonalválasztókat és kapcsolókat. Noha nyílt hálózatmenedzsment protokollok már korábban is léteztek (például az SNMP), és a szabad szoftverek a számítógép-hálózatok üzemeltetésének legalapvetőbb szintjén is megtalálhatóak (gondoljunk csak a ping és a tcpdump parancsokra), a közeljövőben jelentős változásokat hozó OpenFlow protokoll egyedülálló lehetőséget kínál a szabadszoftver-közösségnek, hogy a hálózati infrastruktúra belsejében is kivívja a szoftverek négy alapvető szabadságjogát.

Ez a cikk többek között áttekinti a számítógép-hálózatok és a szabad szoftverek történelmi kapcsolatát, röviden bemutatja egy teljes OpenFlow kontrolleralkalmazás megírásának főbb lépéseit, végezetül ismerteti az OpenFlow körül kialakult ökoszisztéma szabad szoftveres komponenseit.

Tartalomjegyzék

1. Bevezetés

A szabad szoftvereknek a számítástechnika sok területén sikerült teret nyerniük, azonban a nyílt hardverek még csak szűk körben érhetők el. Sőt, manapság még az sem teljesen egyértelmű, hogy ki mit ért nyílt hardver alatt. Mindenesetre számos olyan projekt létezik, amely a közösségi szoftverfejlesztés módszertanát próbálja meg átültetni különböző hardvertervezési projektbe. Ezekből a projektekből mutat be néhány jellegzetes példát a 2. fejezet, kitérve a speciális körülményekből adódó jogi megfontolásokra és licencelési kérdésekre is.

Az OpenFlow protokoll a nyílt hardverek körébe vonja be a hálózati útvonalválasztókat és kapcsolókat. Azonban nyílt hálózatmenedzsment protokollok már korábban is léteztek (például az SNMP), és a szabad szoftverek a számítógép-hálózatok üzemeltetésének legalapvetőbb szintjén is megtalálhatóak (gondoljunk csak a ping és a tcpdump parancsokra). Ennek megfelelően a 3. fejezet áttekinti a számítógép-hálózatok és a szabad szoftverek történelmi kapcsolatát bemutatva, hogy a nagy sebességű adattovábbításért felelős eszközök szoftvereinek módosítására eddig nem volt lehetőség.

Az elmúlt hónapokban nemcsak a szakmai fórumokon, hanem egyéb médiában is számos hír jelent meg SDN (Software Defined Networking) és OpenFlow témában. Az egyik ilyen a Google bejelentése volt, miszerint adatközpontjaikban átálltak az OpenFlow technológiára. A másik érdekes hír arról szólt, hogy egy egyetemről indult, SDN specialista startup céget (Nicira) több mint egymilliárd dollárért vásárolt meg a VMware. Mindezek sejtetik, hogy komoly dolgok történtek/történnek a hálózatos világban, és nagy átalakulás várható akár már a közeljövőben is. A 4. fejezetben egy rövid áttekintést adunk az SDN alapkoncepciójáról és a ma legígéretesebb gyakorlati megvalósításáról, az OpenFlow ajánlásról. A következő, 5. fejezet egy új csomagtovábbítási mechanizmus példáján keresztül veszi sorra egy teljes OpenFlow kontrolleralkalmazás megírásának főbb lépéseit.

Végezetül a 6. fejezet ismerteti az OpenFlow körül kialakult ökoszisztéma szabadszoftveres komponenseit rámutatva arra, hogy a közeljövőben jelentős változásokat hozó OpenFlow protokoll egyedülálló lehetőséget kínál a szabadszoftver-közösségnek, hogy a hálózati infrastruktúra belsejében is kivívja a szoftverek négy alapvető szabadságjogát.

2. Szabad hardver

A szabadszoftver-mozgalom egyik konkrét kiváltó oka az volt, hogy Richard Stallman nem tudta megszerezni a kutatólaborjában lévő nyomtató illesztőprogramjának forráskódját, hogy azt továbbfejlessze [9]. A mozgalom szerint egy szoftver akkor tekinthető szabadnak, ha a felhasználónak joga van (0) bármilyen céllal használni a programot, (1) módosítani a programot, (2) terjeszteni a programot akár pénzért, akár ingyen, (3) terjeszteni a módosított programot. Azért, hogy egy szabad szoftver módosított változata, azaz a szoftverből előállított származékos munka is szabad maradjon, a mozgalom kidolgozta a GPL szoftverlicencet [8]. A licenc úgynevezett copyleft licenc, azaz lehetőséget biztosít a származékos munkák előállítására, de kiköti, hogy a származékos munkákat csak az eredetivel megegyező licencfeltételek mellett lehet továbbadni.

Stallmann a nyomató illesztőprogramjának forráskódja helyett magának a nyomatónak a „forráskódját” is megpróbálhatta volna megszerezni, hogy a hardvert tökéletesítse. A hardverről azonban jóval nehezebb és költségesebb másolatokat készíteni, ennek ellenére a közösségi szoftverfejlesztés sikerén felbuzdulva sokan megpróbálkoztak szabad hardverek fejlesztésével.

A Windowfarms projekt például közösségi alapon gyűjt össze olyan tudásbázist, amely segítségével gyakorlatilag PET palackokból lehet lakásablakokba helyezhető konyhakertet építeni. [10]

A Global Village Construction Set elsősorban mezőgazdasági munka- és erőgépek CAD tervrajzait gyűjti össze [2]. A projekt résztvevői közösen tervezik meg egy önellátó gazdaság működtetéséhez szükséges gépeket.

A nyílt protézis projekt online gyűjtőhelye művégtag-terveknek [3]. A rászoruló vagy egy szakember le tudja tölteni a CAD terveket és elküldheti gyártásra. Azonban a tervek kidolgozása még önmagában nem generál tömegtermelést, így az előállítási költségek nem csökkennek jelentősen. A művégtag előállítását az is nehezíti, hogy a gyógyászati-segédeszközök gyártása szigorúan szabályozott, emiatt a kevés engedéllyel rendelkező szakember között nem alakul ki árverseny.

Szintén jogi akadályai vannak, hogy a nyílt forráskódú GSM bázisállomást bárki üzembe tudja helyezni. Egyrészt a GSM szabvány által használt spektrum szabályozása kormányzati hatáskör. Másrészt a GSM ugyan nyílt szabvány, de használata nem ingyenes. Számos olyan funkció kell a megvalósításához, amelyet még le nem járt szabadalmak védenek. [1]

A nyílt hardverek egyik hátránya tehát az, hogy hiába hozzáférhetőek a hardver előállításához szükséges tervrajzok, maga a gyártási folyamat sokak számára túl bonyolult. Ezen kíván segíteni a kicsit futurisztikus Fab@Home projekt. A projekt 3D nyomtatói egy megadott tervrajz segítségével tetszőleges hardvert le tudnak gyártani, illetve le fognak tudni gyártani. A cél az, hogy egy összetett eszköz innovációjának ne szabjon gátat a gyártás monopóliuma.

Az elektronikai áramkörök tervezéséhez használt Verilog és VSDL hardverleíró nyelvek hiába állnak közel a programozási nyelvekhez, a szokásos szabad licencek helyett speciális licencekkel lehet csak elérni, hogy az ilyen nyelveken megadott szabad hardver származékos munkája is szabad maradjon. A TAPR Open Hardware License például szoftver helyett a dokumentációról és az abból előállítható termékről beszél. A licenc copyleft típusú, tehát származékos munkát is TAPR Open Hardware License alatt lehet terjeszteni. Valamint a licenc a GPL-hez hasonlóan tartalmazhatja, hogy a felhasználónak joga van a licenc későbbi változatára áttérni. Azonban a GPL-től eltérően a származékos munka dokumentációjában egyértelműen meg kell adni, hogy a származékos munka miben tér el az eredetitől.

Összefoglalva látható, hogy többfajta megközelítés létezik a nyílt hardver definíciójához. Nem mindegy, hogy a hardver reprodukálásához használható dokumentációk érhetőek el szabadon, vagy csak a hardver külső vezérlését lehetővé tevő interfészeket hozzák nyilvánosságra.

3. Szabad szoftverek a számítógép-hálózatokban

A hálózatok használatában, üzemeletetésében mindig is jelent voltak szabad szoftverek. A BSD Networking Release 1 volt az első széles körben elérhető hálózati stack. A Usenet és az IRC elosztott rendszerét is közösségi fejlesztésen alapuló szabad szoftve-

rek tartották egyben. Az egyszerű diagnosztikai programoktól (ping, traceroute) a komplexebb forgalomanalizátorokig (tcpdump, Wireshark) terjed azoknak a szabad szoftvereknek a sora, amelyeket a mai napig éles rendszerekben is használnak. A Perl szkriptnyelvet nem csak naplófájlok feldolgozására használják, hanem ahogy mondani szokták, a Perl az a ragasztószalag, amely egyben tartja az internetet. Érdemes lehet még kiemelni az SNMP protokollt használó, link-terhelés monitorozására szolgáló Multi Router Traffic Grapher (MRTG) programot, amely tipikus példája a professzionális eszközökbe is beépített szabad szoftvernek.

Ezzel együtt azonban a hálózati eszközök specializálódásával összhangban a szabad szoftverek kiszorultak a hálózati infrastruktúrából, mivel az egyre nagyobb adatátviteli sebességeket csak speciális hardverekkel lehetett elérni.

4. OpenFlow: az aktuális SDN jelölt

A mai hálózati infrastruktúrával, illetve az azt alkotó hálózati eszközökkel kutatási szempontból az a fő probléma, hogy zárt rendszert alkotnak. A gyártók érthető módon elrejtik a belső, megvalósítási részleteket, ezért a kutatóknak komoly problémát jelent az új hálózati protokollok, innovatív ötletek valós hálózatokban, valós körülmények melletti kipróbálása, tesztelése, mivel nincs lehetőség a belső működést módosítani. Természetesen számítógépekből is lehet teszthálózatokat építeni és új kapcsolási vagy routing algoritmusokat kipróbálni, azonban ezek jóval kisebb működési sebességre és jóval kevesebb port kiszolgálására képesek, mint a valós kapcsolók (switchek), útvonalválasztók (routerek), melyek a legtöbb feladatot hardverből oldják meg.

2. OpenFlow folyamtábla [7]

Erre a problémára ad megoldást az SDN, illetve az OpenFlow ajánlás, amely különválasztja a kapcsolók belső működését (meghagyva a gyártóknak) és a vezérlő logikát, amivel lehetőség nyílik tetszőlegesen programozható hálózatok létrehozására. Tulajdonképpen a jelenlegi eszközön belüli adat- és kontroll sík kerül különválasztásra oly módon, hogy a kontroll rész (vagy annak egy része) kikerül a „dobozból”. Így az egyszerű, hardverből megoldható alapműveletek nagyon gyorsak maradnak, és a gyártóknak nem kell felfedni azok részleteit, míg a vezérlés különválik, és maga a kontrol-

ler nyílt interfészeken keresztül programozhatja a hálózat egészét. Az OpenFlow tulajdonképpen a protokoll a hálózati eszközök és a kontrollerek között.

Innen már csak egy lépés eljutni a hálózati operációs rendszerig, amely például az OpenFlow protokoll segítségével kommunikál a hálózati eszközökkel, és felfele különböző API-kat biztosít a hálózati programoknak, melyek egy magasabb absztrakciós szinten képesek vezérelni a hálózati működést. Ebben az esetben a hálózati alkalmazás vagy szolgáltatás alkotójának nem kell ismernie az alacsony szintű működési részleteket, hanem a hálózati operációs rendszer által biztosított, magasabb szintű programozói környezetben kell dolgoznia, felhasználva az API-kon keresztül elérhető funkciókat.

Az OpenFlow koncepció arra épül, hogy a jelenlegi hálózati eszközök általában úgy működnek, hogy van egy folyamtáblájuk (vagy több ilyen táblájuk), amely különböző információkat tartalmaz arra vonatkozóan, hogy a beérkező csomagokkal milyen műveleteket kell végrehajtania egy eszköznek. Egy útvonalválasztó esetében például ezt úgy kell elképzelni, hogy a beérkező csomag cél IP-címe alapján ki kell választani a megfelelő folyambejegyzést (illeszkedésvizsgálat), és a bejegyzésben jelzett kimeneti porton kell továbbküldeni a csomagot. A táblák feltöltése a kontroll sík feladata, ez hagyományos esetben a hálózati eszközön futó szoftvert vagy szoftvereket jelent. Az OpenFlow ajánlás azzal, hogy kiemeli a kontroll funkciókat az eszközből, a táblák feltöltését, menedzselését egy külön entitásra, a kontrollerre bízza, amit akár mi is létrehozhatunk, és a saját számítógépünkön futtathatunk. A koncepciót a 2. ábra mutatja be. Ennek a megoldásnak az a nagy előnye, hogy ha egyszer már bent van a bejegyzés a megfelelő táblában, akkor a működési sebességet a hardver határozza meg, és a csomagtovábbítás vonali sebességen történik. Ezt egy lehetséges forgatókönyv szerint úgy képzelhetjük el, hogy beérkezik egy csomag a kapcsolóhoz, amely kezdetben nem tartalmaz folyam bejegyzéseket, ezért nem tudja, mit kell tennie. Ezért a kontrollerhez fordul, aki megmondja, mit kell csinálnia, és elhelyezi a vonatkozó bejegyzést a megfelelő táblában. Ezt követően a folyam későbbi csomagjai már nem váltanak ki kommunikációt a kontrollerrel, a megfelelő bejegyzés biztosítja a vonali sebességű működést.

Az OpenFlow lehetővé teszi a csomagfejléc különböző mezőire való illeszkedésvizsgálatot. Ezzel tulajdonképpen egy hálózati rétegeken átívelő megoldást ad, amely például azt is lehetővé teszi, hogy teljesen elfelejtsük a hagyományos IP-t vagy MAC-címeket. Egy OpenFlow bejegyzés az illeszkedési információkon kívül tartalmazza a vonatkozó akciót (pl. továbbítsd a csomagot a 2-es porton), illetve statisztikákat (pl. mennyi csomag illeszkedett eddig az adott bejegyzésre). Ebben a keretrendszerben egyszerűen le tudjuk írni a hagyományos hálózati eszközök működését, például a routingot, a kapcsolást, VLAN-kapcsolást stb.

Az első OpenFlow switch implementációk természetesen szoftverben kerültek megvalósításra. A Stanford Egyetemen készített referencia implementáció például Linux alatt „user space”-ben működik és az operációs rendszer fizikai vagy virtuális interfészeit tudja felhasználni, ezek fogják alkotni a switch portokat. Ennél hatékonyabb megoldást biztosít az Open vSwitch, amely „kernel space”-ben működik, és több operációs rendszer virtualizáló program is támogatja (pl. XEN esetében ez az alapértelmezett virtuális switch, amely a különböző virtuális gépek összekapcsolását teszi lehetővé). Az első hardveres prototípusok megjelenése után számos gyártó előállt a keresk-

delmi forgalomban is elérhető OpenFlow eszközökkel, így ma már a Cisco, HP, Juniper, NEC, stb. mind kínál OpenFlow-képes eszközt különböző kategóriákban.

Természetesen az OpenFlow ajánlás még nem érte el a kész állapotot, jelenleg is folyamatosan fejlesztik, bővítik. A szabványosításra egy külön szervezet (ONF, Open Networking Foundation) jött létre, és az első verziók megjelenése óta számos újabb jelent meg. Ezek egyrészt újabb mechanizmusokat, kifinomultabb műveleteket adnak az eszközhöz, de történtek a hardveres architektúrát is érintő változtatások (pl. pipeline-ba kapcsolt folyamtáblák), illetve a QoS támogatás biztosításához szükséges fontosabb alapelemek is integrálásra kerültek. Fontos megjegyezni, hogy a hardverek jelenleg csupán az 1.0-s verziót támogatják, de a szoftveres verziók már elérhetők az újabb ajánlásokhoz is. Ezenkívül megjelentek olyan generikus, programozható hardverek is, mint például a NetFPGA kártya, amely egy FPGA modul köré épített, PC-be illeszthető kártya, több hálózati csatlakozóval. Bizonyos verziójához például implementálták az OpenFlow működést. Ezekkel az eszközökkel arra is van lehetőség, hogy az új ötleteket közvetlenül hardverre programozzuk le.

5. Új csomagtovábbítási mechanizmus tesztelése

Mint láthattuk, egy kontrolleralkalmazás megírása elegendő ahhoz, hogy egy innovatív ötletet valós számítógép-hálózaton ki tudjunk próbálni. Ez a fejezet egyrészt egy példán keresztül egy rövid betekintést nyújt egy kontrolleralkalmazás elkészítésébe; másrészt megmutatja, hogy mi a teendő akkor, amikor már magát az OpenFlow protokollt is ki kell bővíteni. Azaz mi a teendő abban a szélsőséges esetben, amikor pusztán egy kontrolleralkalmazás elkészítése már kevés egy új ötlet teszteléséhez [5].

(a) *Adattovábbítás MPLS segítségével*

(b) *Adattovábbítás hálózati kódolás felhasználásával*

3. ábra. Többesadás mintatopológiája két forrás és két nyelő esetén

A 3.a. ábrán látható elrendezésnél két adó továbbít adatot két-két vevő felé. Ezt a többesadásos elrendezést MPLS címkéket használó kontrolleralkalmazással meg tudjuk valósítani. Ebben az esetben az s5 és az s6 kapcsolók folyamtábláit kell olyan folyambejegyzésekkel feltölteni, amelyek a források felől érkező adatcsomagokra egy címkét helyeznek. A többi kapcsolóban pedig ezek a címkék azonosítják be a folyamokat, ezeknek a címkéknek a segítségével tudjuk megmondani, hogy a kapcsolók melyik kimeneti portjaira kell az adatcsomagokat továbbítani. A feladatot megoldó kontrolleralkalmazást a NOX keretrendszerben írtuk meg. A többi keretrendszerhez hasonlóan eseménykezelőkön keresztül lehet egy reaktív alkalmazást írni. A 4.a. ábrán látható annak az eseménykezelőnek egy részlete, amelyik egy új kapcsoló észlelésekor fut le. A kódrészlet az s5 kapcsolónak küld le egy folyamtábla-bejegyzést. A bejegyzés a h1 csomópontból küldött és a h3 csomópontnak címzett csomagokra illeszkedik. Az

illeszkedő csomagokra végrehajtandó akciók egy MPLS címkét helyeznek a csomagra, majd a címke értékét egyre állítják (s6 kapcsoló esetén kettes lesz a címke értéke), végezetül kiküldik a csomagot az s7 kapcsoló és az s9 kapcsoló felé.

```
switch (dp_id.as_host()) {
case 5: {
  b = new b_flow_mod();
  b->match_src(h1_ip_addr);
  b->match_dst(h3_ip_addr);
  b->apply_actions()
        ->push_mpls_header()
        ->set_mpls_label(1)
        ->output(s5_port_to_s7)
        ->output(s5_port_to_s9);
  b_send(b);

  // ...

  break;
}
```

(a) egyszerű MPLS továbbítás esetén

```
switch (dp_id.as_host()) {
case 5: {
  b = new b_flow_mod();
  b->match_src(h1_ip_addr);
  b->match_dst(h3_ip_addr);
  b->apply_actions()
       ->push_mpls_header()
       ->set_mpls_label(0)
       ->push_mpls_header()
       ->set_mpls_label_from_counter()
       ->push_mpls_header()
       ->set_mpls_label(1)
       ->output(s5_port_to_s7)
       ->output(s5_port_to_s9);
  b_send(b);

  // ...

  break;
}
```

(b) hálózati kódolás esetén

4. ábra. Kódrészlet egy új kapcsoló észlelésekor lefutó eseménykezelőből

A többi kapcsolóba is hasonló folyamtábla-bejegyzéseket kell adni, azonban az s6 kapcsolót leszámítva az illeszkedési szabályoknál már elég az MPLS címke értékét figyelni.

Figyeljük meg a 3.a. ábrán, hogy az s7 és az s8 kapcsolók közötti linken mindkét adatfolyam forgalmát át kell küldeni. Azonban a dupla adatmennyiség csomagvesztéshez vezet, ha a link kapacitása szűkös. Ilyen esetekben a csomagvesztés elkerülhető az úgynevezett hálózati kódolás (network coding) használatával. A hálózati kódolás itt bemutatott változatában az s7 kapcsoló a két adatfolyam csomagjai helyett a csomagok kizáró vagy (XOR) művelettel előállított változatát küldi az s8 csomópont felé (3.b. ábra). Tehát két bejövő csomagból egyetlen kimenő csomagot állít elő. A végpontok azért tudják megkapni a két adatfolyamot az eredeti formájukban, mert s9 és s10 kapcsolókon egy kódolt és a hozzá tartozó kódolatlan csomag kizáró vagy kombinációja a másik csomagot állítja elő.

A 3.b. ábrán szereplő hálózati kódolást használó elrendezést azonban nem lehet a jelenlegi OpenFlow protokollt támogató kapcsolókkal megvalósítani többek közt azért sem, mert a protokoll nem ismer olyan műveletet, amely két csomag kizáró vagy kombinációját állítja elő. Tehát a hálózati kódolás kipróbálásához először a protokollt kell kiterjesztenünk, majd meg kell írnunk azt a kontrollert, amely felprogramozza a kapcsolókat a 3.b. ábrán látható viselkedésre, végezetül kapcsolókat kell felkészítenünk az új protokollüzenetek lekezelésére.

Az OpenFlow protokollt bővíthetőre tervezték. Van egy külön üzenettípus a kísérleti, úgynevezett Experimenter műveleteknek. A kísérleti üzenetekből lehetnek a szabvány következő verziójában az új szabványos üzenetek, de hogy addig se keveredjenek össze különböző cégek és intézmények kiterjesztései, ezért egy Experimenter azonosítót kell igényelnie a kísérleti üzenetet implementálni kívánó intézménynek a protokoll szabványosítását végző Open Networking Foundationtől. Az egyedi azonosítót fel-

használva azután már mi magunk tudjuk definiálni az általunk használt kiterjesztett protokollüzenetek formátumát és jelentését.

Ha az adatfolyamokat hálózati kódolásnak vetjük alá, akkor a kódolt csomagokban el kell tárolnunk, hogy az adatfolyamok hányadik csomagjait tartalmazza a kódolt csomag. A csomagok sorszámozását az s5, s6 kapcsolókon végezzük el, ahogy a 4.b. ábrán látható kódrészleten szerepel. Az egyszerűség kedvéért a kódolatlan csomagokat is két sorszámmal látjuk el, így a folyamazonosítón kívül még két MPLS címkét adunk a csomaghoz. Az első címke nulla lesz jelezve, hogy a csomag nem tartalmaz információt az ábrán kékkel jelölt adatfolyamból. A második címkét pedig a set_mpls_label_from_counter akció fogja beállítani. Ilyen műveletet azonban nem ismer az OpenFlow protokoll, tehát kísérleti műveletként kell definiálnunk. A kontrolleralkalmazásunkból könnyedén el tudjuk küldeni ezt a kiterjesztett műveletet a kapcsolónak, a kapcsolót azonban csak szoftveres megvalósítás esetén van esélyünk felkészíteni egy kiterjesztett üzenet fogadására. Magának az üzenetnek a feldolgozása nem bonyolult, hiszen csak egy számlálót kell megnövelnünk és a továbbiakban felhasználhatjuk a meglévő set_mpls_label műveletet, azonban a hardveres megvalósítások zártsága miatt erre az egyszerű kiterjesztésre sincs általában lehetőségünk.

A csomagok dekódolása a fentiekhez hasonlóan elvégezhető az s9 és s10 kapcsolókon. A kontroller leprogramozása és a kapcsolók módosítása után a rendszer tesztelését először érdemes egy emulált hálózaton elvégezni. A Mininet képes akár több száz csomópontot is emulálni egyetlen asztali számítógépen [4]. A program a Linux kernel pehelysúlyú hálózati virtualizációjára épül, az emulált hálózat csomópontjaihoz tartozó folyamatokat különböző hálózati névtérben helyezi el. Az emulált hálózati topológiát rugalmasan lehet Python programnyelven előállítani, de magát a Mininetet már nem bővíthetőre tervezték. Persze mivel a Mininet szabad szoftver, elvileg saját igényeinknek megfelelően testre szabhatnánk a forráskódját. De ez karbantartási nehézségekhez vezethet, ha a Mininet egy újabb változatába szeretnénk portolni a változtatásainkat. Jobb megoldást nyújt az úgynevezett monkey patchek használata. Ekkor egy osztály metódusát egy másik fájlból írjuk felül. Ez a megközelítés egy fokkal rugalmasabb az előzőnél, de szintén problémákhoz vezethet, ha egy felülírt osztály interfésze megváltozik.

A Mininet használatának az is az előnye, hogy ezáltal egyetlen virtuális gépen lefuttatható hálózati méréseket, kísérleteket tudunk előállítani és megosztani másokkal. Így könnyen reprodukálható és továbbfejleszthető kutatási eredményeket tudunk létrehozni.

6. Út a szabad szoftveres hálózatok felé?

Az OpenFlow nyílt szabvány, kontroller-keretrendszereknek és szoftveres kapcsolóknak számos szabad megvalósítása érhető el, mégis az egyik jelentős SDN témakörre specializálódott szakportál cikke szerint a nyílt forráskód jelenti a legnagyobb veszélyt az SDN-re [6]. A cikk szerint a fontosabb SDN-es szoftverprojekteket mind elsősorban csak egy-egy cég támogatja, így a projekt fejlődésének irányvonalát és a közösségi együttműködés szabályait is a domináns cég határozza meg. Ráadásul a domináns cég bármikor ki tudja eszközölni ezen szabályok megváltoztatását, ami bármikor bekövetkezhet, ha a céget felvásárolják.

Nyilvánvalóan kockázatos, ha egy vállalat üzleti stratégiája egy olyan nyílt forráskódú szoftverre épül, amely fejlődésére egy másik cégnek jelentős befolyása van.

Azonban a nem nyílt szoftverekkel szemben a vállalat elvileg forkolni tudja a projektet, ha a szoftver fejlődése számára nem megfelelő irányt vesz. Ráadásul, ahogy az 14. ábrán is látszik, nagyjából ugyanarra a feladatra több, egymással versengő nyílt forráskódú alternatíva is létezik. Jóllehet ezek körül a projektek körül általában nem alakult még ki jelentősebb közösség, a projektek néhány fejlesztőjét egyetlen szervezet támogatja. Azonban például az ON.LAB egy olyan fejlesztőlaboratórium, amelyet több nagyvállalat is támogat, tehát közvetlen befolyással egyikük sem rendelkezik az ON.LAB projektjei felett.

14. ábrán látható kontroller-keretrendszereket nem csak az általuk használt programnyelv különbözteti meg. Egy keretrendszer kiválasztásakor fontos lehet még a szoftver licence is. Egy keretrendszerben megírt kontrolleralkalmazás származékos munkának minősül, ezért copyleft alapú licenc esetén az alkalmazást csak a keretrendszerrel megegyező licenccel lehet terjeszteni. Ez alól kivételt jelent a Beacon licence, amelyben külön kitértek arra, hogy a Beaconre épülő alkalmazásokat nem kell származékos munkának tekinteni.

Összefoglalva elmondható, hogy a számítógép-hálózatok ipara egyértelműen a Software Defined Networking irányba halad, azonban az OpenFlow protokoll egyelőre nem hozta el a hálózati eszközök kommodizációját, azaz olcsó OpenFlow kapcsolót még nem lehet kapni. Továbbá hiába nyílt az OpenFlow szabvány, ez nem azt jelenti, hogy valaha is lesz nyílt hardveres kapcsoló. Sőt előfordulhat, hogy a közeljövő nagy tudású kontrollerei sem szabad szoftverek lesznek. Jelenleg azonban annyira fiatal az OpenFlow technológia, hogy lényegében hiányoznak az átütő erejű kontrolleralkalmazások. Így egy vissza nem térő lehetőség kínálkozik a szabadszoftveres közösség számára, hogy például a Linux kernel sikeréhez hasonlóan egy iparág megkerülhetetlen részévé tegye a szabad szoftvereket.

7. Hivatkozások

[1] J. Corbet: *The trouble with OpenBTS*, LWN, February, 2009, http://lwn.net/Articles/320 163/

[2] M. Jakubowski: *Open-sourced blueprints for civilization*, Ted Talk, 2011, http://www.ted.com/talks/marcin_jakubowski.html

[3] J. Kuniholm: *Open Arms*, IEEE Spectrum, March, 2009, http://spectrum.ieee.org/biomedical/bionics/open-arms/0

[4] B. Lantz, B. Heller, N. McKeown: *A network in a laptop: rapid prototyping for software-defined networks*, Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks, ACM, 19, 2010

[5] F. Németh, Á. Stipkovits, B. Sonkoly, A. Gulyás: *Towards SmartFlow: Case Studies on Enhanced Programmable Forwarding in OpenFlow Switches*, ACM SIGCOMM, 85–86, Helsinki, Finland, August, 2012

[6] M. Palmer: *Open Source: The Biggest Risk to SDN*, Blog post, November, 2012, http://www.sdncentral.com/sdn-blog/open-source-the-biggest-risk-to-sdn/2012/11/

[7] R. Sherwood: *An Experimenter's Guide to OpenFlow*, GENI Engineering Workshop June 2010, June, 2010

[8] R. Stallman: *The GNU Operating System and the Free Software Movement*, , Open Sources: Voices from the Open Source Revolution, O'Reilly Media, January, 1999, ISBN 1 565 925 823

[9] S. Williams: *Free as in Freedom: Richard Stallman's Crusade for Free Software*, O'Reilly Media, 2002, ISBN 0-596-00287-4
[10] *Windowfarms project*, http://www.windowfarms.com

A KÖVETŐ KIADVÁNY MARGÓJÁRA – ODM FŐDOKUMENTUMOK WRITERREL

Németh László

Kivonat

Jelen követő kiadvány a korábban használt LaTeX helyett MSZ ISO OpenDocument ODM fődokumentum-formátumban, LibreOffice-szal lett végső formába öntve, kihasználva a LibreOffice LaTeX-ével összemérhető automatizálási és tipográfiai lehetőségeit. Ennek tapasztalatait foglalja össze az írás.

Tartalomjegyzék

1. Bevezetés

A Szabad Szoftver Konferencia és Kiállítás követő kiadványai Zelena Endre tördelő-szerkesztői munkájának köszönhetően (hasonlóan a korábban szintén általa szerkesztett LME konferenciák kiadványaihoz) 2009-ben (1) és 2011-ben (2) a LaTeX keretrendszerrel lettek kiszedve. A 2012-es követő kiadvány – bár félig már elkészült a LaTeX-tördelés – végül OpenDocument ODM fődokumentum-formátumban, LibreOffice-szal tördelve nyerte el végső formáját. A változás hátterében a következők állnak:

- A szerzők részben LaTeX, részben OpenDocument formátumban nyújtották be eddig is cikkeiket.
- A LibreOffice automatizálási és tipográfiai lehetőségei összemérhetők a LaTeX-ével. (3)
- Az OpenDocument formátum MSZ ISO nyílt szabvány, mégis elmarad a használata az egyetemi és közigazgatási szférában a zárt dokumentumszabványokéhoz képest.
- A LibreOffice az OpenOffice.org legsikeresebb utódjának tekinthető. Nagy tudású, platformfüggetlen, kiválóan honosított, ingyenesen is elérhető szabad irodai programcsomag, amelynek alapértelmezett formátuma az OpenDocument, mégis – részben a zárt szabványok említett „népszerűsége” (termékcsapdája) miatt – nem éri el azt az elterjedtséget, mint amit például a nyílt forráskódú böngészőknél mérni.
- A LaTeX szilárdan őrzi a pozícióját a tudományos publikálásban, de terjedése a speciális felhasználói kör miatt korlátozott. Az OpenDocument és a LibreOffice támogatása a zárt szabványok és az ezekre épülő zárt szoftverek nyílt szabványokkal és szabad szoftverekkel való kiváltását jelenti, és ez – tekintettel a zárt megoldások elterjedtségére – kiemelt feladat.
- Az átállás lehetőséget nyújt a LibreOffice kapcsolódó képességeinek megismerésére, ellenőrzésére, dokumentálására (l. ezt a cikket) és a nyílt forráskódnak köszönhetően, akár javítására is, ezzel is elősegítve a szabad szoftverek népszerűsítését.
- A jövőben ugyanúgy megmarad a cikkek LaTeX formátumban való benyújtásának lehetősége a konferencia előadói részére. A LaTeX formátumú cikkek OpenDocument formátumra alakítására külön fejlesztés készült az ODFPy Python programkönyvtár segítségével, amely egy későbbi cikkben kerül ismertetésre.

2. Összefoglalás

A LibreOffice megbízhatóan használható olyan összetett fődokumentumok készítésére, mint a követő kiadvány. A nyílt szabványok számos esetben mutatták fel előnyeiket a követő kiadvány elkészítése során. Például az említett Python fejlesztés mindössze kétszáz sorban a LaTeX dokumentumok címsorait, táblázatait, illusztrációit, az ezekre való hivatkozásokat, lábjegyzeteket, kétféle LaTeX bibliográfiai hivatkozást és egyebeket is ODF-re alakított, a Tralics fejlesztés köztes XML formátumán keresztül. Az eredetileg is SVG formátumban benyújtott illusztrációk nagy előnyének bizonyult (amellett, hogy a LibreOffice támogatja az SVG W3C vektorgrafikus képszabványt), hogy egyszerű szöveges forrásállományukban a betűnevek cseréje a követő kiadványban használt betűk nevére (Linux Biolinum G, Ubuntu, Ubuntu Mono) a beillesztett képeken a betűk cseréjét eredményezte, ezzel különösebb erőfeszítés nélkül javítva a kiadvány megjelenésén.

3. Dokumentumsablonok

Az OTT kiterjesztésű, vagyis szöveges dokumentumsablonok a karakter-, bekezdés-, oldal-, keret- és listastílusok egységesítését teszik lehetővé a dokumentumokban. Ezenfelül a sablondokumentum tartalmazhat készre formázott jegyzékeket, illetve táblázatokat. A címsor-, stb. stílusok beállításával nem kell foglalkozni, ha az új dokumentum adott dokumentumsablon alapján kerül létrehozásra. A jegyzékeket is elég frissíteni. A sablonban lévő táblázat alapján – mivel az OpenOffice.org, LibreOffice nem támogatja az ODF táblázatstílusokat –, új automatikus táblázatformátumot vehetnek fel a felhasználók, amivel lényegesen egyszerűbben formázhatók a táblázatok (bár ez módosíthatóság vonatkozásában nem éri el a táblázatstílusok nyújtotta lehetőséget, egy pár soros makróval könnyedén alkalmazhatók az automatikus táblázatformátumok a dokumentum összes táblázatára, l. (4) Makrók készítése fejezetét).

A dokumentumok stílusai a dokumentumsablon módosításával egyszerűen frissíthetők: a dokumentum betöltésénél a dokumentumsablon módosítását észlelve a program rákérdez a dokumentum frissítésére, illetve fődokumentum megnyitásánál az összes hivatkozott külső dokumentum („aldokumentum") megjelenése egyszerre frissíthető.

3.1. A követő kiadvány sablonja

A konferencia követő kiadványához készült dokumentumsablon a templates.libreoffice.org oldalról tölthető le (5). A sablon a LibreOffice, vagy OpenOffice.org sablonkezelőjével a saját sablonok közé is telepíthető, de közvetlenül megnyitva is szerkeszthető. A sablon szolgál az aldokumentumok és a fődokumentum készítésére is.

A sablon fontosabb megoldásait a mellékelt táblázat foglalja össze. A LibreOffice Graphite betűkészleteivel számos különleges megoldás érhető el, például valódi kiskapitális betű került beállításra a „Linux Biolinum G:smcp=1" bővített betűnévvel a Kivonat címsorban. Kiemelhető még a rejtett Címsor 1 stílusú bekezdések használata a fődokumentum fejezetcímet (cikkcímet) és a szerzőt is feltüntető tartalomjegyzékének automatikus elkészítésére (erre egyszerűbb, de kevésbé automatizált eljárás a tartalomjegyzék utólagos szerkesztése, jelen esetben a cikkcímek kiegészítése a szerzőkkel).

Feladat	Megoldás
Címsorok után legalább három sor maradjon az oldalon.	A Szövegtörzs bekezdésstílusban 3 árvasor beállítása. A Címsor stílusok alapértelmezett szövegbeosztása, az Együtt a következővel szavatolja a feladat megoldását. Ha a címsor utáni bekezdés csak egysoros, és ez éppen az oldal aljára esik, akkor közvetlen formázással ott is beállítandó az Együtt a következővel.
Címsor utáni bekezdés nincs behúzva, a többi igen. Ott árvasor lehet, csak fattyúsor nem.	A címsorok után Szövegtörzs, utána viszont Első sor behúzása stílusú bekezdések következnek. Ezért a Szövegtörzs stílus Szervező lapján a Következő stílus az Első sor behúzása. Mindkét stílusnál 2 fattyúsor van beállítva, ez szavatolja, hogy a bekezdésen belüli oldaltörésnél legalább egy teljes sor átkerül a következő oldalra. Az Első sor behúzásánál viszont nincs bejelölve az Árvasorok kezelése.
Elválasztás bekap-	A Szövegtörzs stílus Szövegbeosztás lapján beállítva. Mivel az El-

Feladat	Megoldás
csolása a Szövegtörzs és az Első sor behúzása stílusokban.	ső sor behúzása a Szövegtörzs származtatott stílusa (l. a Szervező lapján, vagy a Stílusok és formázás ablak hierarchikus nézetét), az elválasztás arra is vonatkozik.
Bekezdésköz nélkül	Szövegtörzs Behúzás és térköz »Bekezdés alatt nullára állításával
Tükrözött margók	Tükrözött oldalbeállítás, a következő margókkal: 3,5, 2,5, 2, 2 cm.
A konferenciakiadvány publikációinak élőfej nélküli új, páratlan oldalon kezdése.	A dokumentumsablon első bekezdése Cím bekezdésstílusú, amely stílus Szövegbeosztás lapján sortörés van beállítva, Első oldal oldalstílussal. Az Első oldal alapbeállítása, hogy az Alapértelmezett oldalstílusú oldal követi (l. a Szervező lapon), páratlan oldalon való kezdéshez az OldalBeállítás listában a Páros és Páratlan helyett a Páratlan oldalbeállításra van szükség.
Cím után szerepelnek a szerzőnevek a cikk elején, de az élőfejben és fő tartalomjegyzékben fordított sorrendben.	Az automatikus megvalósításnál két beviteli mező kéri be a cikk címét és a szerző(ke)t a sablon alapján létrehozott új dokumentumban. A bevitt adatokat tároló felhasználói mezők tartalma „szerző: cím” formában kerül beillesztésre a cím alatt található rejtett Címsor 1 stílusú bejegyzésben. Az elrejtés „csak” háttérszínnel formázott kisbetűs szöveg, hogy annak tartalma utána a fő tartalomjegyzékbe és az Alapértelmezett stílusú oldalak bal élőfejében is automatikusan megjelenhessen hivatkozásként (valódi rejtett címsorok nem kerülnek a tartalomjegyzékbe). A rejtett címsor a véletlen törlés megakadályozására védett szakaszban került elhelyezésre.
Vázlatszintek számozása	A Címsor 1 számozás nélküli címsor, a többi viszont igen, a szinttel megegyező alszintszámmal (l. Eszközök » Vázlatszintek számozása).
Különböző élőfej a páros és páratlan oldalakon.	Kétféle megoldás is lehetséges: a Páros és Páratlan oldalstílusok váltakoztatása a Szervező lapjaik megfelelő beállításával, vagy egy oldalstílus, jelen kiadványban az Alapértelmezett használata a Formátum »Oldal »Élőfej »Páros és Páratlan oldal tartalma azonos jelölőnégyzet kikapcsolásával. (Ez utóbbi esetben a páros oldali élőfej formázása nem mindig frissül annak módosítása után, ilyenkor az állomány újbóli betöltése segít.)
A cikk első Címsor 2 stílusú címsora új oldalon és pontosan az oldal tetején kezdődjön.	Ezekre nincs közvetlen automatizálási lehetőség. A sablonban a tartalomjegyzék utáni üres bekezdés formázása tartalmaz egy bekezdés utáni oldaltörést. Az ilyen töréspontok után viszont az új oldalon kezdődő bekezdés stílusnak megfelelő bekezdésköze nem tűnik el az oldal tetején, így azt közvetlen formázással kell nullára állítani, hogy ugyanazt a hatást érjük el, mint normál (nem közvetlen formázással kapott) oldaltörésnél.
A cikk tartalomjegyzéke ne tartalmazza a Címsor 1 stílusú címsort.	A cikk tartalomjegyzékében alapértelmezés szerint megjelenik a cikk címét tartalmazó Címsor 1 stílusú bejegyzés is, ami csak a konferenciakötet fő tartalomjegyzékében kell, hogy megjelenjen. Ennek törlésére, illetve a megfelelő stílus beállítására (1) a Jegyzék beszúrása párbeszédablak Bejegyzések lapján az első szint

Feladat	Megoldás
	minden eleme törlésre került, (2) a Stílusok lapján a Szint 1-hez egy új, „Láthatatlan” bekezdésstílus került hozzárendelésre, amely 2 pontos betűformázással a Címsor 1 számára kihagyott függőleges helyet csökkenti le kisebbre, (3) valamint itt a Szint 2-höz lett hozzárendelve a Tartalomjegyzék 1 bekezdésstílus a Tartalomjegyzék 2 helyett, a Szint 3-hoz a Tartalomjegyzék 2, és így tovább a felhasznált szintekig.
Listastílusok	A magyar tipográfiának megfelelő Lista 1 és Számozás 1 listastílusok tabulátor helyett szóközzel választják el a jelet, illetve számot a szövegtől. A számok ezért jobbra igazítottak, a behúzás pedig olyan mértékű, hogy a kétjegyű számok se lógjanak ki a bal szöveghatáron. Ez a 0,5 cm-es behúzás megegyezik az „Első sor behúzása” stílus behúzásával is.
Kis és nagy táblázatok tipográfiája	A sablonban szereplő táblázat sorai a bekezdésen belül is megtörhetnek oldaltörésnél. Kisméretű táblázat a táblázat kijelölése és a Beszúrás »Keret segítségével keretbe helyezhető. A keret bekezdéshez horgonyozva, és az oldal szövegtartalmához pozicionálva lehetővé teszi, hogy a táblázaton belül ne legyen oldaltörés, az átfolyó szöveg mégis egyenletesen töltse ki az oldalt. A táblázathoz felirat a Beszúrás »Felirat... menüponttal adható, ami a Táblázat »Táblázat tulajdonságai »Szövegbeosztás » Együtt a következő tulajdonságát is bekapcsolja. Ez utóbbi nagyobb, többoldalas táblázatok esetén azt eredményezi, hogy a táblázat hibásan új oldalon kezdődik, ezért ilyenkor ki kell kapcsolni.
Lábjegyzet-elválasztóvonal törlése	A lábjegyzet feletti vonal törölhető az oldalstílusok Lábjegyzet lapján az Elválasztó vonal hosszúságának 0%-ra állításával.
Valódi méretezett betűk a lábjegyzetszámozásban	Valódi méretezett apró betűk a Linux Biolinum G és Libertine G betűk „sups” Graphite betűtulajdonságával kapcsolhatók be a Lábjegyzet-horgony és Lábjegyzet-karakterek stílusban. (4)
Jobbra igazított lábjegyzet-számozás	A Lábjegyzet-karakterek „algn” Graphite tulajdonságának beállításával: Linux Biolinum G:sups=1&pnum=1&algn=1
Nagyobb térközök az Előformázott szöveg stílusú bekezdések szélein, nem közvetlen formázással, hanem stílusból	Az Előformázott szöveg (vagy egyéb azonos, például listák) stílusú bekezdések előtt és után nagyobb térközök készíthetők az új „Ne tegyen térközt az azonos stílusú bekezdések közé” bekezdésformázással, így a Bekezdés felett és alatt megadott értékek csak a határon fognak életbe lépni. Az új funkció (amelyet korábban a szegélyek belső margójával és Egyesítés a következővel tulajdonságával lehetett kiváltani) viszont a bekezdésstílus beállító ablakában még nem érhető el, csak a közvetlen Bekezdésformázás ablakban. A megoldás a stílus Szervező lapján az Automatikus Frissítés bejelölése, és utána a bekezdés közvetlen beállítása.
Automatikus névelők a címsorok előtt.	A LibreOffice Graphite betűkészleteinek „arti” betűtulajdonságának beállításával (4), vagy a tipográfiai eszköztár (6) ikonjával formázhatók úgy a számhivatkozások, hogy a megfelelő („a” vagy „az”) névelő jelenjen meg előttük automatikusan.

4. Fődokumentumok

A fődokumentumok több szöveges dokumentum egy kiadványban való összeállítására szolgálnak. Az ISO ODM fődokumentum-formátum kezelése során külön szakaszként kerülnek betöltésre a külső dokumentumok (aldokumentumok). A dokumentumok oldalszámozása, hivatkozásai a fődokumentumban elfoglalt helyüknek megfelelően módosul, miközben az eredeti aldokumentum állományok nem változnak. A fődokumentum további szöveggel, közös címlappal, jegyzékekkel egészíthető ki.

A fődokumentum lehetővé teszi, hogy a szerzők és a lektorok önállóan, kényelmesen és biztonságosan dolgozhassanak egy nagy dokumentum fejezetein, jelen példában egy tanulmánykötet tanulmányain. A fődokumentum és részeinek egységes megjelenéséhez az is szükséges, hogy a fő- és az aldokumentumok ugyanazon dokumentumsablon alapján készüljenek. Az ugyanazon dokumentumsablonhoz való kötés utólag, az aldokumentum szövegének új, sablon alapján létrehozott üres dokumentumba való kézi másolásával, vagy a Template Changer (7) OpenOffice.org, illetve LibreOffice kiegészítővel is megvalósítható.

4.1. A követő kiadvány fődokumentuma

A következő táblázat foglalja össze a LibreOffice fődokumentum-kezelésével kapcsolatos feladatokat, problémákat és megoldásukat.

Feladat	Megoldás
Fődokumentum létrehozása adott sablon alapján	A kívánt sablon megnyitása, majd a Fájl »Küldés »Fődokumentum létrehozása menüpont kiválasztása. (Ha a sablon nem üres, akkor azt több aldokumentumra szedi szét a kiválasztott stílus alapján a LibreOffice a fődokumentum számára). Ezek után megadható a fődokumentum neve. A fődokumentumhoz a megjelenő Navigátor (F5 gombbal előhívható) Beszúrás ikonmenüjének Fájl menüpontjával adható hozzá új aldokumentum.
Új oldalstílus a címlaphoz és egyéb oldalakhoz	A Fődokumentum Címoldal oldalstílusa az Első oldallal szemben láblécet nem tartalmaz, valamint nem csak páratlan oldalon kezdődhet, így a címlaphoz, belső borítóhoz, szennycímlaphoz. kolofonoldalhoz is megfelelő.
Hivatkozások helyreállítása	Az aldokumentumok beszúrása, teljes frissítés során a hivatkozások helyén a „Hiba: A hivatkozás forrása nem található” üzenet jelenhet meg. A fődokumentum mentésével, ismételt betöltésével a hivatkozások helyreállíthatók.
Jegyzékek frissítése	Míg az oldalszámok automatikusan frissülnek a fődokumentumban, a jegyzékek, mint jelen esetben a tartalomjegyzékek oldalszámozásának frissítése külön kéréssel lehetséges: a Navigátor fődokumentum nézete Frissítés ikonmenüjének Jegyzékek menüpontját kell kiválasztani a fődokumentum megnyitása után.
Egyedi irodalomjegyzék-hivatkozások a cikkekben	A LibreOffice saját irodalomjegyzék-kezelője a jegyzékek frissítésénél összevonja az aldokumentumok irodalomjegyzék-hivatkozásait, így a fődokumentumba egy közös irodalomjegyzék illeszthető be. Az összevonást elkerülendő, a jegyzékek közös frissítése helyett csak a tartalomjegyzék frissítését szabad elvégezni,

Feladat	Megoldás
	például a következő makróval (a makrót kétszer futtassuk le egymás után, hogy valóban helyes oldalszámokat kapjunk!) : Sub UpdateTOC x = ThisComponent.DocumentIndexes For i = 0 to x.Count - 1 If InStr(x(i).Name, "Tartalomjegyzék") or InStr(x(i).Name, "Table of Contents") Then x(i).Update Next End Sub
Keretstílus beállítása (keretszegély eltüntetése)	A sablonban a Keret keretstílus szegélye kikapcsolásra került, hogy az aldokumentumok közvetlenül formázott kereteinek ne kerüljön visszaállításra a szegélye a fődokumentumban.
Oldalhoz horgonyzott képek helyes megjelenítése	Oldalhoz horgonyzott képek hibásan jelennek meg a fődokumentumban. A megoldás a képek bekezdéshez horgonyzása, viszont a kép oldalhoz, illetve az oldal szövegtartalmához való pozicionálása.

5. Irodalomjegyzék

(1) *Szabad Szoftver Konferencia és Kiállítás 2009*, FSF.hu Alapítvány, 2009, http://konf.fsf.hu/2009/szszkonf2009_konferencia_kiadvany.pdf

(2) *Szabad Szoftver Konferencia és Kiállítás 2011*, FSF.hu Alapítvány, 2011, http://konf.fsf.hu/2011/szszk2011_koveto_kiadvany_v_2.1.pdf

(3) Németh László, *LibreOffice – Úton a DTP felé*, In: Szabad Szoftver Konferencia és Kiállítás 2011, FSF.hu Alapítvány, 2011

(4) Németh László, *Kiadványszerkesztés LibreOffice Writer szövegszerkesztővel*, http://www.numbertext.org/libreoffice, 2011

(5) http://templates.libreoffice.org/template-center/szabad-szoftver-konferencia, 2013

(6) http://extensions.libreoffice.org/extension-center/typography-toolbar, 2013

(7) http://extensions.libreoffice.org/extension-center/template-changer, 2013

ÉPÍTSÜNK SZUPERSZÁMÍTÓGÉPET SZABAD SZOFTVERBŐL!

Őry Máté

KIVONAT

A műszaki területen dolgozó kutatóknak gyakran van szükségük nagy számítási teljesítményre. Még jobb, ha ezt könnyű használatba venni, és gyorsan átfutnak a feladatok. Az elmúlt nyáron a BME-nek lehetősége volt egy klaszter rendszerű szuperszámítógép beszerzésére, melynek számítási teljesítményére a kutatók már a próbaüzem során lecsaptak. A rendszer kialakítását házon belül oldottuk meg, szabad szoftverekre támaszkodva. Ennek kapcsán röviden bemutatom a nagy teljesítményű számításokra használt technológiát, majd megosztom a Műegyetemen kialakított rendszer működését és tapasztalatainkat.

Tartalomjegyzék

1. A szuperszámítógépek

1.1. Áttekintés

A szuperszámítógép egy olyan számítógépes rendszer, amely a beszerzése idejében szokásos számítógépeknél 2–4 nagyságrenddel nagyobb számítási teljesítményre képes, amit párhuzamosítással, több feldolgozó egység összehangolt működtetésével ér el.

A szuperszámítógépek alkalmazását, valamint a velük foglalkozó területet HPC-nek (high performance computing 'nagy teljesítményű számítás') nevezzük.

A gépeket elsősorban tudományos célra használják, párhuzamosítható lebegőpontos számításokra. Ilyen igény nagy számban fordul elő a fizika, kémia és biológia területén. Ezen kívül HPC rendszereket alkalmaznak bankok és biztosítók gazdasági számításokra, filmgyárak grafikai munkára, és különböző iparágak a műszaki tervezést segítő szimulációkra. A nagyhatalmak kormányai és hadseregei is igénybe veszik a rendszereket.

Fontos kiemelni, hogy a HPC területéről jelentős fejlesztések „szivárogtak" át a fogyasztói piacra. Jó példa erre a napjainkban már a mobiltelefonokban is megtalálható, az első szuperszámítógépek alapját adó FPU (floating point unit 'lebegőpontos egység'), a szuperszámítógépekben később gyakran használt vektorprocesszor működési elvén alapuló GPU (graphical processing unit 'grafikus feldolgozóegység'), vagy a többmagos processzorok esetében is alkalmazott SMP (symmetric multiprocessing 'szimmetrikus párhuzamos feldolgozás') architektúra. Nem csak a véletlennek köszönhető ez: a műszaki korlátok átlépésének igénye folyamatosan jelentkezik ebben a szegmensben.

A világban működő legerősebb szuperszámítógépeket minden évben kétszer számba veszi a TOP-500.org projekt. [11] A rangsorolás alapja a LINPACK teszt eredménye, az R_{max} érték. [12] A teszt egy elosztott mátrixinvertálási feladat futásidejéből számol effektív GFLOP/s értéket. Egy FLOP/s alatt másodpercenként egy végrehajtott lebegőpontos számítást értünk (floating point operations per second 'lebegőpontos művelet másodpercenként'). A nagyságrendeket mutatja be az 1. és 2. táblázat.

Év	Gép	R_{max}
1976.	Első igazi szupergép (Cray–1)	80 MFLOP/s
1993.	Első TOP500 listaelső (CM-5)	59 GFLOP/s
2001.	Első magyar szupergép (NIIF)	60 GFLOP/s
2012.	BME szupergép	6 TFLOP/s
2011.	Új NIIF szupergépek összesen	40 TFLOP/s
2012.	TOP500 lista első (Cray Titan)	18 PFLOP/s

1. Szuperszámítógépek a történelem során

Gép	CPU-mag	GPU	R_{max}	Lemez	Memória	Ár (forint)
Cray Titan	229e	18,6e	17,6 PFLOP/s	10 PB	710 TiB	44 mrd.
NIIF szsz.	5,5e	0	40 TFLOP/s	2,5 PB	20 TiB	1 mrd.
BME szsz.	372	4	6 TFLOP/s	40 TB	1,5 TiB	70 millió
HP ProBook	2	0	6 GFLOP/s	250 GB	4 GiB	160 ezer

2. Napjaink számítógéprendszereinek jellemző teljesítménye

Összehasonlításként egy 2012-es felső kategóriás mobiltelefon CPU-ja 200 MFLOP/s, egy felső kategóriás PC processzor 100 GFLOP/s körüli teljesítményre képes. Egy felső kategóriás videokártya 3 TFLOP/s körüli számítási teljesítményt hirdet, viszont ez a többi adattal szemben – a tudományos alkalmazások számára kisebb jelentőséggel bíró, csupán egyszeres pontosságú (32 bites) lebegőpontos számokra vonatkozik, míg a dupla pontosságú (64 bites) műveletekben ennek a teljesítménynek a tizedét sem nyújtja.

1.2. Párhuzamos feldolgozási architektúrák

A nagy számú processzor együttműködésének megvalósítására néhány, egymástól kisebb-nagyobb mértékben eltérő megoldás alakult ki az ezredforduló környékére.

5. A NUMA architektúra

6. Az SMP architektúra

SMP (Symmetric Multiprocessing 'szimmetrikus párhuzamos feldolgozás') megoldás esetén egy operációs rendszer kezel több CPU-t, és ezek egy belső buszon osztott operatív memóriát használnak (14. ábra). A megoldás korlátja az osztott memóriához való hozzáférés skálázhatósága. A gyakorlati tapasztalatok szerint 32-64 magnál jelentősen többet nem lehet ilyen módon hatékonyan kezelni. Ez a megoldás önmagában ma már nem jellemző a szuperszámítógép kategóriában.

A NUMA (Non-Uniform Memory Access 'nem azonos memóriaelérés') architektúra esetén még nagyobb számú CPU-t kezel egyetlen operációs rendszer (5. ábra). A megoldás lényege, hogy az egyes CPU-k dedikált memóriát érnek el közvetlenül, azonban (lassabban) a rendszer összes memóriáját meg tudják címezni. A NUMA HPC kategóriában jelenleg az SGI Ultraviolet a legnagyobb szereplő. Kisebb méretű NUMA megoldásnak tekinthető az Intel jelenleg használatos, több processzort kezelni képes QPI buszon alapuló technológiája is.

7. A cluster architektúra

A cluster 'fürt' architektúra több, egymáshoz fizikailag közel elhelyezett, saját operációs rendszert futtató számítógépből álló rendszer, melynek elemeit (compute node 'számítási csomópont') gyors hálózat köti össze. (7. ábra). Az ilyen rendszerekre általában egyetlen számítógépként tekintünk, amelynek node-jait csak egy fejgépen keresztül használjuk, külön-külön nem.

Az MPP (Massively Parallel Processing 'erősen párhuzamos feldolgozás') a cluster olyan változata, ahol a gépek közti hálózati összeköttetés a szokásosnál sokkal sűrűbb,

valamint a számítási csomópontok száma is nagy. A kategóriában jelenleg az IBM Blue Gene a meghatározó szereplő.

Napjainkban a felsorolt hagyományos megoldásokat jelentős mértékben felváltották kombinált változataik.

8. A fat cluster architektúra

Az új clusterek nagy része jelenleg *fat cluster* 'kövér/vastag fürt'. Ez nagyobb SMP gépekből alkotott clustert jelent (8. ábra).

9. A GPU-val gyorsított cluster

Jelentős teret kapott a GPGPU-k (General Purpose Graphical Processing Unit 'általános célú grafikus processzor') használata is. Elhelyezésük egyik véglete az, amikor a cluster egyes csomópontjaiba 1-1 kártyát helyeznek a CPU műveleteinek gyorsítására

(9. ábra). A másik végletben egy SMP rendszerbe 8-12 kártyát helyeznek el (10. ábra). Ilyenkor a számítás döntő részét a kártya végzi.

10. A GPU cluster

Meg kell említeni még a hagyományos szuperszámítógépek mellett a GRID rendszereket is, amelyek egy lazán csatolt cluster architektúrájára emlékeztetnek. Két fő ága van a GRID világnak: Az infrastruktúra GRID kifejezetten erre a célra használt, földrajzilag távol lévő gépekből álló cluster (például EGEE, NorduGrid). A desktop vagy önkéntes GRID pedig egyéb célra használt gépek szabad erőforrásait hasznosítja (például SETI@home, LHC@Home). A földrajzi távolság áthidalására internetet használnak. Ennek a megoldásnak a hátránya a nagy adminisztrációs teher, amelyet GRID middleware-ek készítésével próbálnak megoldani, valamint az, hogy a gépek között nincs, vagy lassú a kommunikációs lehetőség, így csak a nagyobb független részfeladatokra osztható problémák oldhatóak meg vele hatékonyan.

1.3. Interconnect

A hagyományos clusterek esetében a gépek között közvetlen üzenetváltásra van lehetőség, amelynek késleltetése és átviteli sebessége jelentősen befolyásolja számos probléma hatékony megoldását. Ezt a clusteren belüli kapcsolatot a HPC zsargon interconnectnek nevezi.

A megfelelő interconnect kiválasztása és megtervezése fontos optimalizálási feladat a szuperszámítógép építése során. Míg a gyorsabb megoldások jelentős költséget jelentenek, elégtelen összeköttetés esetén a csomópontok idejük jelentős részében az i/o műveletekre várnak.

Az egyik legelterjedtebb megoldás az Infiniband. Legújabb változatának (FDR) hasznos átviteli sebessége 40 Gb/s, míg két gép közti késleltetése 1 µs alatti. A rendszer azonban talán a legdrágább kereskedelmi forgalomban kapható korszerű megoldás. A technológia alapja ugyan nyílt szabvány, de jelenleg csak két gyártó foglalkozik vele. Piaci helyzetükből és a kis számú felhasználóból adódóan a szoftvertámogatás nem tökéletes, hiába nyílt forrású a jelentős része.

A kisebb kommunikációs igényű számításokhoz ideális választás lehet a gigabit

Ethernet, amely ma már gyakorlatilag minden nagyobb teljesítményű számítógépben gyári felszereltség. Ingadozó és nagy késleltetését kedvező ára és szoftveres támogatottsága ellensúlyozza.

A két lehetőség között helyezkedik a lassan a HPC világban is teret hódító 10 gigabit Ethernet. Bár az Infinibandnél ára kedvezőbb, teljesítménye jelentősen elmarad mögötte.

Az üzemkritikus rendszerekben az interconnect infrastruktúrát is redundánsan, minden eszközt duplázva telepítik. Ilyenek a katonai rendszerek, vagy például a naponta többször modellszámításokat végző – pénzügyi, meteorológiai területen működő gépek. Ezzel szemben a számítási teljesítményre optimalizált rendszerekben csak a leggyakrabban elromló alkatrészek (táp, lemezek) redundánsak.

2. Szuperszámítógépek Magyarországon

2.1. NIIF Intézet

Magyarországon az első szuperszámítógép az NIIF Intézet (Nemzeti Információs Infrastruktúra Fejlesztési Intézet) Victor Hugo utcai géptermében működött 2001-től. Ennek a gépnek a bővítések ellenére a számítási teljesítménye az évtized végére elenyészővé vált. Az intézetnek lehetősége volt EU-s projekt keretében a konvergencia-régiókban szuperszámítógép vásárlására, beüzemelésére. [13]

A projekt keretében nyugat-európai szinten is versenyképes teljesítményű rendszert sikerült felépíteni a magyar kutatók számára. Három kutatóegyetemen különböző architektúrájú rendszereket állítottak üzembe.

Debrecenben egy 18 TFLOP/s számítási teljesítményű, cluster (MPP) rendszerű gép működik (Xeon processzorok, SGI gyártmányú, Infiniband interconnecttel ellátott cluster). Szegeden egy ehhez hasonló, HP gyártmányú, Opteron processzoros fat cluster üzemel, 14 TFLOP/s teljesítménnyel. Pécsen az SGI ccNUMA rendszerű gépe működik, amely 10 TFLOP/s -ra képes. Az NIIF Intézet budapesti géptermében is működik egy, a szegedihez hasonló, de kisebb teljesítményű gép.

Szegeden az elmúlt hetekben üzembe helyeztek két új node-ot, amelyekben 6-6 darab GPGPU kártya található. Ezzel a rendszer teljesítménye közel 6 TFLOP/s -szal nőtt.

A rendszert kiegészíti a gépek közelében, valamint Sopronban és Dunaújvárosban elhelyezett komplex storage szolgáltatás (PB-os nagyságrendben). A négy gép az NIIF Intézet országos gerinchálózatára csatlakozik.

2.2. BME

2012 nyarán a BME-nek is lehetősége volt egy cluster beszerzésére. Ez az első olyan gépünk, amit szuperszámítógépnek nevezhetünk. A rendszer célja az NIIF Intézet által nyújtott szolgáltatások kiegészítése. A házon belüli megoldás használatba vétele a kutatóknak sokkal egyszerűbb, és a desktop gépeket már kinőtt feladatokat gond nélkül, minimális várakozással lehet futtatni. Felhasználóink beszámolói szerint az új gépen néhány óra alatt lefutó feladatokat korábban hetekig futtattak PC-n.

A projektek 60 évnyi CPU-időt használtak az első 3 hónapban, átlagosan 70%-os kihasználtság mellett.

Az R_{max} = 6 TFLOP/s teljesítményű rendszer egy fejgépből, valamint 30 darab

2×6 magos számító csomópontból áll. A 30 node-ból kettőben 2-2 darab Nvidia Tesla M2070 GPGPU is található. [14]

2.3. További szupergépek

Az országban különböző kutatóintézetekben és egyetemeken is működik cluster rendszerű HPC számítógép. Ezt mutatja be 3. táblázat. Magánszektorban működő rendszerekről nem sok publikus adatot találni.

Hely	R_{max} TFLOP/S	CPU/GPU	Gyártó	Hálózat	Beszerzés
ELTE	3,5	Xeon	HP	Gbit Ethernet	2009.
OMSZ	14	Xeon	IBM	Gbit Ethernet	2010.
Győr	?	Xeon+Tesla	Vegyes	Infiniband	2010–
KFKI	5	Xeon	SGI	Infiniband	2010.
Miskolc	2,9	Xeon	HP	Infiniband	2011.
BME	6	Xeon+Tesla	HP	Infiniband	2012.

3. Clusterek a magyar állami intézményeknél

Különböző projektek keretében néhány TFLOP/s összteljesítményű GRID rendszerek is működnek például a KFKI-nál, a Debreceni Egyetemen, az NIIF Intézetnél, valamint a SZTAKI-nál és a BME-n is.

3. Szoftverkörnyezet

3.1. Operációs rendszer, rendszerbeállítások

Akadémiai környezetben terjedt el a HPC, ebből adódóan mindig is jó pozícióban voltak a UNIX rendszerek. Manapság a Linux jár az élen; a TOP500 listán 3 Windowsost kivéve minden rendszer UNIX-szerű.

Nem ilyen szabad a választás a disztribúció tekintetében. A szoftver- és hardvergyártóknak hála szinte csak a RHEL (Red Hat Enterprise Linux) és a SLES (SUSE Linux Enterprise Server) használata jellemző. Érdemes még megemlíteni a GRID rendszereken elterjedt, CERN által fejlesztett hírhedt RHEL-klónt, a Scientific Linuxot.

A BME szuperszámítógép szoftveres környezetét magunk alakítottuk ki. Az egyes rendszerbeállításokra nincsen általánosan elfogadott álláspont, optimalizálásuk műterheléses méréssel elvileg sem lehet sikeres. Így az elméleti okoskodáson kívül a tapasztalatok és a saját mérések tudnak csak segíteni.

Első ilyen kérdés a Hyper-Threading volt. Kikapcsolását a méréseink is igazolták: a gépünkön futó számítások szinte kivétel nélkül lebegőpontosak, így a sok területen jellemző i/o intenzív vagy ingadozó terhelésű használattal ellentétben a beállítás kikapcsolásával jelentősen gyorsabb futást érhetünk el alkalmazásainkkal.

A hasonló rendszerekhez képest kis tárolási kapacitású rendszert szereztünk be. Nem i/o intenzív használatot, és ezen belül is főleg szekvenciális írást-olvasást remélve végül a hardveres RAID6-ot választottuk, ezzel viszonylag rossz véletlenszerű írási teljesítményt vállalva a kapacitás hatékony kihasználásáért.

A választott ütemező működéséből adódóan megosztott fájlrendszerre van szüksé-

günk. A 10Gbit Etherneten működő NFS elvárásainkat teljesítette, így más megoldásoknál sokkal egyszerűbb beüzemelése miatt mellette döntöttünk.

Kialakítottunk egy menedzsmenthálózatot a hálózati eszközöknek, valamint a HP kiszolgálókon elérhető iLO felületnek (Ethernet). A fájlrendszer elérése, és az általános célú hálózati forgalom a 10Gbit Ethernet hálózaton történik. A HPC számítások üzenetei az Infiniband QDR hálózaton közlekednek.

3.2. Rendszerindítás

A clusterek általában lemez nélküli gépekből állnak, hálózatról bootolnak. A gépekben van egy kisebb lemez, de ezeket szinkronban tartani nehéz. Emellett RAID használatára sincs lehetőség, ami a rendelkezésre állást jelentősen csökkenthetné.

Ezen érveket összegezve a hasonló rendszerekben szokásos hálózati, PXE rendszerindítást választottuk.

Ennek kialakításához a Dracut initrd-infrastruktúrát bővítettük. [15] A / fájlrendszer egy csak olvasható NFS megosztás a fejgépen, azonban egyes fájlokat írhatóvá kell tenni a rendszer helyes működéséhez. Ennek eléréséhez a node-ok első indításakor lemásoljuk a szükséges fájlokat egy-egy gépspecifikus könyvtárba. Ezeket írható módon osztjuk meg. A kialakított init folyamat így minden gépen az egységes / katalógust, valamint a gép egyedi fájljait tartalmazó megosztást csatolja, és az írható fájlokat a linuxos `mount` parancs `--bind` opciójával csatoljuk a helyére. Ezzel a módszerrel transzparens módon tudunk kötetek közötti hivatkozásokat létrehozni.

Ezen kívül hozzá kellett adni a 10Gbit Ethernet kártyánkhoz szükséges kernelmodult, valamint egy, az ideiglenes adatoknak és a swapnek helyet adó belső lemezt indításkor ürítő, szükség esetén particionáló programot.

3.3. Ütemezés

Az interaktív futtatás a számítási teljesítmény kihasználása szempontjából nem hatékony. (Erre hivatkozva tiltották ki annak idején a programozókat a gépteremből.) Mivel a szabad erőforrásoknál több a számítási igény, a megfelelő, automatikus ütemezés feltétlenül szükséges a hatékony kihasználáshoz. Ne felejtsük el, hogy bár az operációs rendszerek alapvetően képesek a rendelkezésre álló kapacitásnál több feladatot egyszerre (időosztásosan) futtatni, ez jelentős többletterhet ró a gépre.

Ennek megoldására a feladatokat megfelelő sorrendben egymás után elindító operátor munkájához hasonló elvek mentén készült batch 'kötegelt' ütemezőket használunk.

Egy ilyen rendszertől elvárjuk azt is, hogy olyan módon ütemezzen, ami lehetővé teszi kevesebb vagy több gépen futó, illetve rövidebb vagy hosszabb várható futási idejű feladatok vegyes indítását is. Olyan rendszert keresünk, aminek az ütemezési stratégiáját a felhasználók is igazságosnak érzik: rövid várakozási idővel induljanak el az új feladatok akkor is, ha egy másik felhasználó sok, vagy hosszú feladatot küldött be. Mindezt persze úgy kell megvalósítani, hogy a gép kihasználtsága is megfelelően alakuljon.

A legtöbb batch ütemezőre elmondható, hogy működésük egy felhasználó szemszögéből a következő: A felhasználó beküld egy feladatleírót, majd amint lehet, az ütemező futtatja a megadott parancsot, végül ha elkészült, értesíti a felhasználót.

A clusterekben a Sun Grid Engine (ma: Oracle Grid Engine, valamint forkja az

Open Grid Scheduler), valamint a SLURM szoftverek terjedtek el erre a célra. Korábbi pozitív tapasztalataink, valamint a felsorolt elvárásoknak való jobb megfelelősége miatt egy HTCondor rendszert üzemeltünk be.

3.4. HTCondor

A HTCondor meglehetősen kiforrott szoftver – 1988 és 2012 között Condor néven fejlesztették, az utolsó bő évtizedben Apache License alatt. Alapvetően GRID jellegű rendszerek menedzselésére szolgál, azonban ez a heterogén környezethez való alkalmazkodás alkalmassá tette más, például cluster rendszerek üzemeltetésére is. A Wisconsini Egyetemen egy csoport aktívan fejleszti, valamint a (főleg akadémiai) felhasználók is sok kódot adományoznak. Közösségi támogatása is jó.

Nincsenek különböző várakozási sorok és ezekhez rendelt gépek: a szabad erőforrások és a feladatok hirdetéseket (ClassAds) adnak föl, ezeket párosítja az ütemező. A felhasználó az összes elvárását feltételként adhatja meg, az erőforrások kiválasztását rangsorolási szempontokkal befolyásolhatja. A szoftverrendszer az erőforrás-használat monitorozását, felügyelését és a „számlázást" is megoldja. Az ütemezési stratégiája igazságos, mivel egyetlen sok feladatot beküldő felhasználó nem lehetetleníti el mások számára feladatok futtatását, mindemellett a tapasztalatok szerint gazdaságosan használja ki a rendelkezésre álló erőforrásokat.

Az ütemező a frontend gépen fut, itt lehet a feladatot beküldeni, ezeket előkészíteni (például interaktívan fordítani). A felhasználók csak ide tudnak belépni SSH kapcsolat létesítésével.

Minden gépről kérhetőek egyesével magok, vagy az egész gép. A Tesla kártyát tartalmazó csomópontokra, valamint a fejgépekre csak rövid futásidejű CPU alapú feladatok kerülnek, így elkerülhető az, hogy azért ne lehessen GPGPU-s feladatot futtatni, mert azon a gépen éppen napok óta egy egyszálú feladat fut. A fejgépen azért engedtük meg néhány perces feladatok futását, hogy a csomópontok teljes terhelése esetén is lehessen tesztelni a beküldendő programokat.

Az 1 órán keresztül kihasználatlan gépeket lekapcsoljuk, ezzel az eddigi mérések alapján százezer forintos nagyságrendű megtakarítást érünk el évente a villanyszámlán. Ha egy új feladat elvárásai nem elégíthetőek ki a szabad bekapcsolt gépekkel, a szükséges számú csomópontot elindítjuk a HP iLO SSH-s interfészén keresztül.

A felhasználók a tapasztalatok szerint feladatonként legalább egy teljes gépet kérnek, így a memória allokálása nem szükséges – teljes gépek használatánál ez nem is lehetséges. A futásidőkorlát kötelező megadása javítaná a hatékonyságot. A jelenlegi megoldás szerint a „túlfutott" feladatokat kilőjük. Ha nincs ilyen korlát megadva, akkor az operátor nem tud megkülönböztetni egy beragadt és egy hosszasan számoló esetet.

3.5. Párhuzamos programozás

A különböző feladatok más megközelítést igényelnek. Clusterek több csomópontján való párhuzamos futás elérésére elterjedt megoldás az MPI (Message Passing Interface), valamint ritkábban elődje, a PVM (Parallel Virtual Machine) használata is előfordul. Párhuzamos, többszálú programok írását segíti az OpenMP vagy a Pthreads. A két megközelítés együttes használata is gyakran előfordul.

Más problémák könnyen feloszthatóak több független feladatra. Ezek jellemzően a

parameter sweep 'paramétersöprés' kategóriába esnek, ahol ugyanazt a feladatot kell elvégezni különböző paraméterekkel. Népszerű megoldás ilyen problémákra a Hadoop.

A GPGPU programozására kétféle megoldás jellemző: az Nvidia Cuda, valamint a gyártófüggetlen OpenCL.

Négyféle MPI-implementációt, kétféle Cudát és más egymással nehezen összeegyeztethető technológiákat támogatunk (a felhasználók megdöbbentően változatos binárisokat tudnak hozni).

Ezen környezetek támogatásához egy Environment Modules nevű, több mint 20 éves TCL szkriptet használunk. [16]Ez a megoldás különböző futtatási környezetek közti választáshoz ad egy egyszerű interfészt. Így megbízhatóan lehet lokális alapértelmezett MPI-implementációt, Cuda-verziót választani egy-egy parancs kiadásával.

3.6. Adminisztráció

Három hónap alatt 40 projektben dolgozott az egyetem hat karján működő 15 különböző tanszék közel száz kutatója, valamint egyes projektekben külső kutatók is részt vesznek. Ilyen szerteágazó felhasználói körnek nagy értékű eszközhöz jogosultságok kiosztásához megfelelő szintű autentikációt magunk nem tudtuk vállalni, különösen azt figyelembe véve, hogy nem minden kutató hajlandó ezért személyi igazolványával elzarándokolni az operátori helyiségbe.

Ilyen körülmények közt adta magát, hogy az eduID föderációhoz csatlakozzunk. [17] Ez a rendszer a magyarországi felsőoktatási és kutatóintézmények felhasználóinak az elosztott azonosítását valósítja meg.

A felhasználók ezzel a megoldással egy direkt a szupergéphez létrehozott, Django alapú webalkalmazásba tudnak belépni. Itt lehetőségük van a rendszer dokumentációjának megtekintésére, valamint projektek javaslására és kezelésére.

A webes felületen javasolt projekteket egy bizottság elbírálja, majd kedvező döntés esetén gombnyomásra létrejön az új témaszám. Az accountot egy Python kód hozza létre a Django rendszer modellje alapján. A projekt összes tagja egy közös projekt-felhasználó nevében lép be, a saját SSH kulcsával. A projektigényléskor becslést kérünk az erőforrás-használatra. A megadott értékek közül a lemezhasználatot ki is kényszerítjük a kvóta alrendszerrel.

A kezdő cluster-felhasználók első próbálkozása sok esetben az, hogy örömükben az új accounttal belépnek a fejgépre, és elindítanak rajta egy nagy programot. Az ilyen, a fejgép használhatatlanságát okozó magatartásoknak kiszűrésére a PAM limits rendszer hatékonynak bizonyult.

A felhasználók mind a 30 node-on léteznek: az NFS-ben azonos `/etc/passwd` fájlt találnak, de nem tudnak interaktívan belépni az SSH alkalmas beállítása miatt. Nem láttuk szükségét elosztott felhasználókezelés megvalósítását, ez a sokkal egyszerűbb megoldás is megfelel az elvárásoknak.

4. Köszönetnyilvánítás

A rendszer kialakítását *dr. Szeberényi Imre* vezetésével, *Guba Sándor*ral végeztük a BME Közigazgatási Informatikai Központban. A vasat a BME TIO munkatársai üzemeltetik.

A gép beszerzését is tartalmazó program az „Új tehetséggondozó programok és ku-

tatások a Műegyetem tudományos műhelyeiben" TÁMOP - 4.2.2.B-10/1-2010-0009 című projekt támogatásával valósul meg.

5. Hivatkozások

[11] The Top500 List, http://www.top500.org/
[12] Jack J. Dongerra, *Performance of Various Computers Using Standard Linear Equations Software*, Manchester, 2012.
[13] Mohácsi János, *A HBONE+ projekt áttekintés*, Budapest, 2012, http://www.hboneplus.hu/node/160
[14] BME Szuperszámítógép, https://superman.eik.bme.hu/
[15] Dracut, https://dracut.wiki.kernel.org/
[16] Environment Modules Project, http://modules.sourceforge.net/
[17] EduID, http://www.eduid.hu/

MIRAGE

Páli Gábor János
pgj@bsd.hu

Kivonat

A Mirage egy OCamlben írt, exokernel alapú, nyílt forráskódú kísérleti operációs rendszer, amely biztonságos, nagy teljesítményű hálózati fejlesztésre kínál megoldást. Ennek segítségével az alkalmazások kódja tetszőleges POSIX operációs rendszer felett fejleszthető és tesztelhető, majd egyetlen kapcsoló átállításával mindez lefordítható egy Xen felett futó, specializált mikrokernellé. Mivel a Xen számos cloud szolgáltatás alapját képzi, ezért a Mirage lényegében egy egyszerű, az eddigi megoldásoknál biztonságosabb és jobban kezelhető alternatívát valósít meg. Ezt és ennek fejlesztés alatt álló FreeBSD portját mutatjuk be röviden ebben az írásban.

Tartalomjegyzék

1. Bevezetés

A Mirage biztonságos, nagyteljesítményű, cloud és mobil platformokon futó hálózati alkalmazások fejlesztésére alkalmas kísérleti operációs rendszer, amelyet a Cambridge University Computer Laboratory-ban fejlesztenek. A projekt célkitűzése, hogy a gyakorlatban is alkalmazható, hatékony és könnyen kezelhető eszközt ajánljon fel mindazoknak, akik szeretnének megbízható módon specializált operációs rendszereket készíteni.

A projekt komolyságát mutatja egyébként, hogy a saját honlapja [1] is egy ilyen alkalmazásként fut, rendelkezik egy komplett (IPv4) TCP/IP hálózati stackkel, amelynek felhasználásával a fejlesztői készítettek teljes értékű SSH és DNS szervereket is, a C implementáció méretének mintegy töredékéből.

1.1. Az alkalmazás mint operációs rendszer

A rendszer alapvetően exokerneles [2] felépítésű, amelynek lényege, hogy lehetőleg minél kevesebb absztrakciót kényszerítsenek a fejlesztőkre, ezáltal lehetőség nyílik szinte teljesen saját megközelítésükben felépíteni az alkalmazásaikat. Ezért az ilyen típusú megoldások leginkább csak egyszerűbb kényelmi szolgáltatásokat nyújtanak, például a rendszerben fellelhető erőforrások elérését teszik lehetővé, de a hozzájuk kapcsolódó protokollokat illetően az alkalmazás szabad kezet kap. Gyakran hivatkozzák emiatt az ilyen módon elkészített alkalmazásokat „library operating system” [3] néven.

A fejlesztés során az alkalmazások kódja tetszőleges POSIX-kompatibilis operációs rendszeren, például FreeBSD vagy Linux alatt megírható és kipróbálható, amely aztán a futtató rendszer lecserélésével lefordítható egy önálló, specializált miniatűr operációs rendszerré. Ilyen miniatűr rendszer lehet például egy Xen hypervisor felett futtatható mikrokernel, de a megoldás absztrakt felépítésének köszönhetően tulajdonképpen sokféle formában előállítható.

A Xen választását ebben az esetben valószínűleg az indokolja, hogy ez szintén a Cambridge-i Egyetemről került ki, és az ott dolgozó kutatók a lehetséges alternatívák közül ezt ismerik a legjobban. Emellett a Xen definiál egy kényelmes hardverfüggetlen interfészt, amelyre anélkül tudunk alkalmazásokat illeszteni, hogy el kellene vesznünk a hardverek megszólításával és zökkenőmentes üzemeltetésével kapcsolatos problémákban. Másrészt a Xen napjainkban egy igen népszerű ipari virtualizációs megoldássá nőtte ki magát, ezért számos cloud szolgáltatás megvalósításának alapját képezi.

Amikor viszont ilyen virtualizált alapokra építünk szolgáltatásokat, akkor akaratlanul is további rétegeket halmozunk fel a rendszer felépítése során: szükségünk van egy hagyományos, általános célú operációs rendszerre, különböző (DNS, SSH, HTTP, stb.) szerverekre, esetleg ezek felett további bővítési lehetőségekre, (Python, PHP, Ruby, stb.) szkriptelési lehetőségekre és így tovább. Ez a megszokott „LAMP” („Linux, Apache, MySQL, PHP”) konfiguráció, amely ugyan sokak számára könnyen kezelhető, a megvalósítás szemszögéből jelentős pazarlás és bonyolítás egyszerre, amely jelentős biztonsági kockázatokkal jár [4].

2. A funkcionális nyelvek felbukkanása

Ezért a Mirage kidolgozóinak egyik alapvetése, hogy valamilyen módon csökkenteni kell tudnunk rendszerünk rétegezettségét és bonyolultságát is, miközben megtartjuk annak rugalmasságát, javítjuk a teljesítményét és megbízhatóságát. Erre a szerepre a funkcionális programozást, azon belül pedig az OCaml programozási nyelvet [5] választották.

A funkcionális programozási paradigma ugyanis már a kezdetektől absztrakt, matematikai gondolkodás (ún. lambda-kalkulus) mentén épül fel, lehetővé téve olyan szintű specifikációk megfogalmazását és futtatását, amelyek erős garanciákat adnak helyes és összefüggő programok kidolgozására, ezáltal eleve megbízhatóbbak. Másrészt, absztrakt jellegüknél fogva ilyen típusú nyelvek mögött nagyon hatékony és intelligens futtatórendszerek és fordítóprogramok állnak, amelyek nagy mértékben tudják segíteni a programozó munkáját. Továbbá ezekben a nyelvekben meglehetősen elfogadott módszer különböző, csupán egy adott szakterület fogalmaival dolgozó, nem általános célú programozási (al)nyelvek („domain-specific language", DSL) létrehozása. Ennek köszönhetően egy magasabb szemantikai szinten fejezhetjük ki a programjainkat, így a fordítóprogram nagyobb optimalizációs szabadsággal rendelkezik, illetve a nyelv tervezői logikailag ki tudják zárni a helytelen programokat a fordításból.

A funkcionális nyelvek világán belül többféle megközelítés létezik a fentebb említett tulajdonságok teljesítésére, amelyek értelemszerűen eltérő kompromisszumokkal járnak az implementáció és a kifejezési készség tekintetében. Ezek közül testesít meg egyet az OCaml, amely erős statikus típusozású nyelv. Az erős típusozás itt azt jelenti, hogy a háttérben nem történik automatikus implicit konverzió az egyes alkalmazott típusok közt, hanem a programozónak mindig egyértelműen jeleznie és kérnie kell ezeket. Ezzel elkerülhetőek például az implicit konverzióból fakadó kellemetlen meglepetések, de egyúttal segítik a programozóban is tudatosítani az adatáramlást. A statikus típusozás pedig arra utal, hogy a fordításkor (tehát lényegében csupán a program szövegét nézve) tisztában kell lennünk azzal, hogy melyik ponton melyik változó milyen típusú értéket képvisel. Ez pedig azért hasznos, mert így a program szerkezete automatikusan ellenőrizhetővé válik, és segít rávilágítani a benne rejlő hibákra.

Ebben a paradigmában jellemző lehet még a mellékhatások erős elszigetelése és a végtelen adatszerkezetek támogatása is (ld. például a Haskell nyelvet), viszont ez jelentős mértékben el tudja bonyolítani a futtatáshoz szükséges támogatás megvalósítását. Az OCaml ezért ez utóbbi lehetőségekhez nem nyújt közvetlen támogatást, cserébe viszont egy egyszerű futtatórendszerrel rendelkezik, valamint a fordító által előállított kód sebessége sem marad el túlságosan a C nyelvű változatától. A típusok állandósított jelenléte egyébként messze nem zavaró az erősen statikus típusos funkcionális nyelvekben, ugyanis a fordító képes magától kikövetkeztetni a programbeli nevekhez tartozó típusokat és csak az olykor előforduló félreértések tisztázása esetén (vagy dokumentációs céllal) kényszerül a programozó ezeket megadni. Ennek köszönhetően viszont képes felvenni a versenyt a dinamikus nyelvek megszokott kényelmével.

2.1. Funkcionális hálózati programozás – egy példán keresztül

Tehát itt lényegében OCamlben írt funkcionális programok lesznek azok az alkalmazások, amelyeket aztán operációs rendszerként futtathatunk. Mögöttük természetesen egy komoly modulrendszer áll, ahol implementálták többek közt a hálózati kapcsolatok kezelését, így nekünk, az alkalmazás fejlesztőjének ezekre csak hivatkoznunk kell.

Példaként tekintsük a következő kódrészletet. Ez egy egyszerű, TCP-n keresztül kommunikáló echo alkalmazást valósít meg, amely a 8081 `tcp4` porton várja a klienseket, és a tőlük kapott adatokat visszaírja nekik.

```
open OS
open Net.Nettypes

let echo () =
  Manager.create (fun mgr interface id ->
    let src = None, 8081 in
    Flow.listen mgr (`TCPv4 (src,
      (fun (addr, port) t ->
         Console.log (sprintf "From %s:%d" (ipv4_addr_to_string addr) port);
         let rec loop () =
           lwt res = Flow.read t in
           match res with
           | None ->
              Console.log "Connection closed";
              return ()
           | Some data ->
              Flow.write t data >>=
              loop
         in
         loop ()
      )
    )))

let _ = Main.run (echo ())
```

OCamlben a `let` (és `lwt`) kulcsszó segítségével tudunk neveket, tehát változókat vagy függvényeket definiálni. Ekkor lényegében a definíciók sorrendjüknek megfelelően kiértékelődnek és létrehozzák a nekik megfelelő típusú objektumot. A felső szinten megadott `let` definíciók az egész modulban látszanak, viszont a bennük szereplő `let` definíciók csak az ezeket befoglaló egységben, ezért itt a nevek után még egy, hatáskört hozzákötő `in` kulcsszó megadása is kötelező.

Másik fontos kulcsszavunk a `fun`, amely névtelen (avagy lambda-) függvények definícióját vezeti be. Ezek tulajdonképpen paraméteres blokkoknak tekinthetőek, és felhasználásukkal egy-egy törzset tudunk hozzárendelni a függvényekből képzett utasításainkhoz. (A névtelen függvényekkel kifejezett részleteket természetesen nevesített alfüggvény formájában is megadhattunk volna, de ez funkcionális nyelvekben lényegében szükségtelen, amikor csak egyetlen helyen használjuk fel.)

Ilyen speciális utasítás lesz tehát például a `Manager.create`, amely felépíti az alkalmazás számára a hálózati kapcsolatot és ezután meghívja a paraméterként megadott blokkot a beállított paraméterekkel. Ezt felhasználva a másik speciális utasítás, a `Flow.listen` nyit egy TCP (v4) kapcsolatot a 8081-es porton, ahol az adatok feldolgozását és generálását egy újabb paraméteres blokkban írjuk le.

A belső blokkunkban megkapjuk paraméterként a küldő címét és portját, valamint a kapcsolat állapotát. Ennek alapján tudunk adatot beolvasni a küldővel a háttérben kialakított TCP-csatornán keresztül a `Flow.read` függvény meghívásával. Ezután a konkrét eredmény alapján tudunk arról dönteni ún. mintaillesztés (`match` .. `with`) segítségével, hogy valóban kaptunk-e adatot, vagy sem. Amennyiben nincs már több

adat (vagyis `None` értéket kaptunk vissza), akkor befejeződik a feldolgozás, ha viszont érkezett valamennyi (`Some`) adat, akkor a mintaillesztés segítségével kiemeljük, és a `Flow.write` függvénnyel visszaírjuk a küldőnek. Utána pedig folytatjuk egy, a funkcionális nyelvekből jól ismert rekurzív hívással. Itt a `loop` belső függvény saját magát hívja meg, miközben a `>>=` (bind) operátorral egyúttal átadja a belső állapotot.

Végezetül a `_` névtelen függvény definíciójával azt kérjük a rendszertől, hogy értékelje ki a `Main.run` függvényt, amely paraméterül megkapja a fentebb ismertetett függvényünket, vagyis a futtatandó alkalmazást.

3. Röviden az implementációról

Miután elkészítettük az alkalmazásunkat, többféle módban tudjuk fordítani. Az egész módszer lényege, hogy az OCaml programokat nem fordítjuk le teljesen, hanem csak egy futtatórendszer nélküli tárgykódot állítunk elő, tehát a programunk még linkelés előtt álló változatát. Az OCaml fordító ilyenkor kiválogatja a programunkban hivatkozott modulokat és függvényeket, és csak azokat teszi bele a tárgykódba, amelyeket valóban használunk is. Ennek köszönhető az, hogy az alkalmazásunk mellé a Mirage beépített elemei közül mindig csak a szükségesek kerülnek bele a mikrokernelbe.

Ezt egyébként a következő módon érhetjük el:

```
$ ocamlopt -output-obj -o app.o echo.ml
```

Ezt követően már csak annyi a teendőnk, hogy kiválasszuk a megfelelő backendet, vagyis a platformot, ahol futtatni akarjuk a programot. Ennek különböző szintjei lehetnek:

- A Xen virtuális interfészére támaszkodva, paravirtualizált kernelként futtatjuk. Ilyenkor a működéshez szükséges elemeket, mint például a hardvereszközök vagy a memóriaelérést javarészt egy leegyszerűsített rendszerrel, a „Xen MiniOS” [6] egy módosított változatának felhasználásával érjük el. Ilyenkor az alkalmazásunk a komplett memóriát egyben, 64 biten éri el. (Ezzel később még foglalkozunk.)
- Egy meglevő POSIX rendszer alrendszereire támaszkodunk, és az Ethernet forgalmat TUN/TAP alkalmazásával megcsapoljuk. Ilyenkor a hálózati kártyát állományhoz hasonlóan kezeljük, ahova írva Ethernet kereteket tudunk küldeni, olvasva pedig fogadni. Ennek nyilván a közvetlen Xen feletti futáshoz képest nagyobbak a költségei, azonban az alkalmazás viselkedése így könnyebben nyomon követhető.
- Egy POSIX rendszer hálózati alrendszerére támaszkodunk, és a kommunikációt egy TCP/UDP socketen keresztül bonyolítjuk le. Ez az előbbinél egy lassabb módszer, szintén a kezdeti prototípus elkészítésekor érdemes leginkább használni.

Természetesen ezen kívül még további backendek is léteznek, a Mirage futtatható akár WebSockets felett az `ocamljs` fordítónak köszönhetően, de akár a Google AppEngine vagy Android platformokra is telepíthető az `ocamljava` és az OCaml fordító ARM backendjén keresztül. Bizonyára érezhető, hogy egy magasabb szintű programozási nyelv ilyenkor sokkal könnyebben kezelhető, hiszen a fejlesztőnek ekkor nem kell az egyes platformok eltéréseivel foglalkozni, hanem elegendő csupán a programlogikát leírnia, és a megfelelő modulok elvégzik a feladat többi részét.

3.1. Futtatás Xen felett

Mielőtt azonban még továbbhaladnánk, egy kicsit térjünk vissza a natív, vagyis a Xen feletti futtatáshoz. Azért érdemes erre még időt és helyet szánnunk, mert az itt kialakított rendszer igyekszik minél jobban idomulni az OCaml futtatórendszer igényeihez és működéséhez. Ez leginkább a memóriakezelésben mutatkozik meg, mivel hasonlóan a többi funkcionális nyelvhez, itt is automatikus szemétgyűjtögetés történik a program futtatása során. Ez pedig egy operációs rendszer szerves részeként alapjaiban meghatározza annak teljesítményét.

Az ezzel kapcsolatos hiedelmek és panaszok ellenére ez alapvetően egy remekül működő módszer, amellyel a programozót meg lehet szabadítani az erőforrás-kezeléssel járó problémáktól és ezáltal kiküszöbölni az ebből fakadó hibalehetőségeket. Fontos azonban hozzátenni, hogy ez funkcionális nyelvekben azért működhet valóban, mert a programokban a memóriakezelés teljesen implicit marad, a programozónak nem kell tudnia annak jelenlétéről, hiszen az adatszerkezeteket is absztrakt matematikai eszközök, ún. algebrai adattípusok segítségével építheti fel.

11. ábra. A Mirage memóriakezelése Xen felett

Az OCaml szemétgyűjtögetője egyébként az adatokat két osztályba sorolja: a rövidtávú információkat az ún. minor heapbe rendezi, a hosszútávúakat pedig a major heapbe. Emiatt a minor heap tulajdonképpen a verem fogalmához áll közelebb, viszont jóval általánosabb, és szinte teljesen automatikusan, a felhasználás és viselkedés alapján kerül eldöntésre, hogy melyik objektumok landolnak itt. Emellett a major heap pedig nagy lépésekben, akár 4 MB-os superpage-enként is képes nőni, ezzel mérsékelve a működéshez szükséges lapok számát és azok adminisztrációjának költségét, amely így javulást eredményez a teljesítményben is.

A másik fontos tényező, hogy a hálózati (és általánosságban az I/O) műveleteknek is minél kevesebb költséggel kell járniuk. Erre a Mirage az „IO page" nevű absztrakciót használja, amely tulajdonképpen a külvilággal történő kommunikáció alapköve. Egy ilyen page egy OCaml tömböt rejt maga mögött, amely tetszőleges memóriabeli helyre mutathat. Ezt természetesen az alkalmazás fejlesztője nem tudja módosítani, úgy

látja, mintha az valamilyen programbeli számítás eredménye lenne. Ez lehet memória leképezett I/O terület, vagy akár a hálózati kártya által a memóriában valahol elhelyezett adat.

Az OCaml tömbökből pedig lehet kérni különböző nézeteket, altömböket, amelyek az eredeti tömb valamelyik szeletét ábrázolják – szintén tömbként. Ez a tulajdonság kapóra jöhet akkor, amikor például hálózatról érkező kereteket akarunk feldolgozni, hiszen így folyamatosan el tudjuk hagyni a csomag már feldolgozott részeit, miközben a háttérben valóban csak egy mutatót állítgatunk. Az IO page-ek emiatt alkalmasak a C nyelvi implementációban megjelenő mbuf(9) adatszerkezet működésének utánzására, vagyis általuk egy ún. másolás nélküli („zero copy") csomagfeldolgozást tudunk megvalósítani.

12. ábra. Pehelysúlyú kooperatív szálak futtatása

Mindezek mellett lehetőségünk van szálak használatára is az alkalmazásokban, amelyek segítségével logikai szinten fel tudjuk osztani a programot különböző párhuzamosan futó folyamatra. A szálak létrehozását és kezelését az `lwt` (mint „lightweight threading") nevű, szintén OCaml nyelven írt könyvtár végzi [7]. Ez a könyvtár kooperatív szálkezelést valósít meg, vagyis nem található benne dedikált ütemező, a szálak egymásnak önkéntesen adják át a vezérlést, például akkor, amikor várakoznak valamilyen adatra. A szálak törzsét egyébként ilyenkor monadikus blokkok formájában fogalmazzuk meg, amelyeket futás közben össze tudunk fésülni más hasonló blokkokkal. Noha a monádok, a monadikus programozás inkább a Haskell funkcionális nyelvben terjedt el, OCamlben is gyakran alkalmazzák, hogy tisztább, matematikailag jobban kezelhető programokat kapjanak. Ez a típusú szálkezelés jobban kiszámítható, ám nem jelent túlzottan nagy mértékű visszaesést a teljesítményben.

Az iméntiekből leszűrhető, hogy igazából egyetlen Mirage kernel sem valósít meg igazi párhuzamosságot, amikor a szálak ténylegesen egymással egy időben futnak. Ennek a kérdését ismét a gazdakörnyezetre, vagyis jelen esetben a Xenre bízza, és ha erre van szükségünk, akkor egyszerűen csak több ilyen rendszert kell elindítanunk felette. Ekkor a különböző Mirage példányok osztott memória vagy adatbázisok (pl. XenStore) segítségével tudnak kommunikálni egymás között. Ez a megközelítés vi-

szont talán annyiban hasznos, hogy így az egyes funkciók teljesen elszigetelhetőek egymástól, a rendszer egyes részei akár külön újra is indíthatóak.

4. Mirage/kFreeBSD

Végezetül néhány szóban bemutatnám az idén nyáron tett, kéthónapos cambridge-i tanulmányutam eredményeit, amelynek keretében be tudtam kapcsolódni a Mirage fejlesztésébe. Ennek célja a FreeBSD kernel port fejlesztése volt, ugyanis a Mirage fejlesztői mellett több FreeBSD fejlesztő is a Computer Laboratory munkatársaként dolgozik, sőt, azonos projektben dolgoznak. Ennek következtében szinte elkerülhetetlen volt, hogy a két technológiai irányt ne egyesítsék egy ilyen kísérletben, hiszen a FreeBSD mint az egykori egyetemi UNIX, a BSD reinkarnációja, önmaga remek táptalaj az operációs rendszerek témakörében tett tudományos kirándulásoknak.

A kísérlet elsődleges célja tehát az, hogy a Mirage ne csak Xen, hanem a FreeBSD kernel felett is tudjon futni. Pillanatnyilag ezt úgy oldjuk meg, hogy az alkalmazásokból kernelmodulokat készítünk, ezáltal a kernel címterében és privilégium szintjén fog futni az alkalmazás. Természetesen mint hagyományos POSIX rendszer, FreeBSD felett egyszerű OCaml alkalmazásként TUN/TAP vagy socketek segítségével enélkül is tudunk Mirage programokat futtatni, itt a cél most ezek kernelbe történő beemelése.

Emellett továbbá célunk az is, hogy a FreeBSD ipari minőségű hálózati stackjét le lehessen cserélni a Mirage saját stackjével. Vagyis hogy képesek legyünk egy OCamlben kifejlesztett hálózati motort futtatni a FreeBSD kernel belsejében, ezzel pártatlanul le tudjuk mérni, és össze tudjuk hasonlítani a két megközelítést. A Mirage-ot ugyanis érték olyan kritikák, amelyek szerint a benne leírt specializált alkalmazások tartalmazhatnak olyan egyszerűsítéseket, amelyek révén gyorsabbnak bizonyulnak a tesztek során. Helyette célszerű lenne tehát minél szabályosabb keretek közé szorítva megvizsgálni a működését (13. ábra).

Ehhez fel kell építeni a kapcsolatot a rendszerben található hálózati eszközök és az OCaml kód között. A FreeBSD kernel ugyanis tartalmazza a megfelelő meghajtókat, amelyek a hálózatról érkező adatokat mbuf(9) formájában egy érkezési sorba pakolják, valamint hasonló stílusban továbbítják egy küldési sorból. Ezek eléréséhez viszont elsőként meg kell találnunk a rendszerben fellelhető hálózati interfészeket, amelyekre aztán a betöltött kernelmodul rá tud telepedni és a C hálózati stack elől kiemelni az Ethernet keretek tartalmát.

Erre az `ether_input()` függvényen keresztül van lehetőségünk, amely minden beérkező keret esetén meghívódik a meghajtó és a keret adataival paraméterezetten. Hasonló módon az `ether_output()` függvény alkalmas arra, hogy rajta keresztül szabályosan megszerkesztett mbuf(9) adatokat küldjünk. Erre ideiglenes jelleggel a netgraph(3) könyvtárat használtuk, de folyamatban van az áttérés a pfil(9) eszközeire, mivel az tisztább felületet biztosít ehhez.

A többi funkció esetében csupán arra volt szükség, hogy a modulhoz kapcsolt OCaml futtatórendszer tudja használni a kernel által felkínált `malloc()` és `free()` függvényeket, valamint üzeneteket megjeleníteni a konzolon és elérni a rendszerórát. Hogy a betöltött modul a rendszer többi részével együtt, a háttérben tudjon futni, a futtatandó kód egy megfelelően létrehozott kernelszálra kerül (`kthread_add()` és `kthread_exit()` függvények).

A legtöbb erőfeszítést talán a megfelelő fordítási konfiguráció megtalálása, és az ehhez szükséges Makefile állományok elkészítése jelentette. A FreeBSD kernel ugyan-

13. ábra. *A Mirage/kFreeBSD backend felépítése*

is igen szigorúan bánik mind a betölthető modulok fajtáival (nem tartalmazhatnak például pozíciófüggő kódokat (PIC)), mind az FPU használatával (alapvetően nincs rá támogatás), miközben az OCaml alkalmazások ezt adottnak tekintik. Olyannyira, hogy az OCaml alapkönyvtárában az időt is valós számként ábrázolják, illetve a szemétgyűjtögető is lebegőpontos értékek felhasználásával határozza meg a működéséhez szükséges paramétereket.

Mindezek miatt tehát várhatóan további fejlesztésekre lesz szükség ahhoz, hogy valóban használható FreeBSD kernel portot kaphassunk, noha az elsődleges eredmények eléggé biztatóak. Reményeink szerint a Mirage hamarosan megjelenik a FreeBSD Portgyűjteményében, ahonnan a többi porthoz hasonlóan, minden nehézség nélkül telepíthetővé, és ezáltal szélesebb körben elérhetővé válik.

5. Összefoglalás

A Mirage egy nagyon izgalmas, rengeteg kutatási potenciált magában hordozó projekt, amely az elkövetkezendő években várhatóan tovább fog fejlődni, és némi szerencsével például a Xenhez hasonló ipari szintű eszközzé növi ki magát. Örömömre szolgált, hogy belekóstolhattam a fejlesztésébe, és ezen keresztül lehetőségem adódott megtapasztalni egy régi vágyamat, vagyis operációs rendszer fejlesztését funkcionális nyelven – mindezt Cambridge festői szépségű városában.

6. Hivatkozások

[1] http://openmirage.org/

[2] Engler, Dawson R.; Kaashoek, M. Frans; O'Toole Jr., James (1995). *Exokernel: An Operating System Architecture for Application-Level Resource Management.* Proceedings of the fifteenth ACM symposium on Operating systems principles, pp. 251-–266.

[3] Porter, Donald E.; Boyd-Wickizer, Silas; Howell, Jon; Olinsky, Reuben; Hunt, Galen (2011). *Rethinking the Library OS from the Top Down.* Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems.
[4] Madhavapeddy, Anil; Mortier, Richard; Sohan, Ripduman; Gazagnaire, Thomas; Hand, Steven; Deegan, Dim; McAuley, Derek; Crowcroft, Jon (2010). *Turning down the LAMP: Software Specialisation for the Cloud*. 2nd USENIX Workshop on Hot Topics in Cloud Computing HotCloud 2010.
[5] Doligez, Damien; Frisch, Alain; Garrigue, Jacques; Rémy, Didier; Vouillon, Jérôme (2012). *The OCaml system release 4.00 – Documentation and user's manual*. Institut National de Recherche en Informatique et en Automatique.
[6] http://wiki.xen.org/wiki/Mini-OS
[7] http://ocsigen.org/lwt/
[8] https://github.com/pgj/mirage-kfreebsd

LIBREOFFICE A NEVELÉSTUDOMÁNY SZOLGÁLATÁBAN[5]

Pénzes Dávid

KIVONAT

Az alábbi szöveg célja, hogy bemutassa miként lehet igényes, gyorsan tördelhető, tudományos folyóiratot szerkeszteni LibreOffice-szal; miként segíti a magyar fejlesztésű, kiegészítőként letölthető Lightproof és Linux Libertine G egy folyóirat megjelenését, miként segíti egy új folyóirat születését a szabad szoftveres ökoszisztéma.

Remélhetőleg ez a rövid szöveg is bizonyítja, hogy nem kell méregdrága szöveg- és kiadványszerkesztőket beszerezni ahhoz, hogy igényes, magas szakmai követelményeknek megfelelő folyóiratot készíthessen egy közösség.

Tartalomjegyzék

[5] Jelen szöveg nem feltétlenül tükrözi a Neveléstudomány folyóirat szerkesztőségének véleményét, kizárólag a szerzőét.

1. Bevezetés

Sokszor hangzik el – főleg „külsős”, hozzá nem értő szájából, – hogy az OpenOffice.org nem jó, nem alkalmas semmire, maximum gépelésre. Ez a sztereotip hozzáállás legfőképpen abból fakad(t), hogy az OpenOffice.org fejlesztése „viszonylag lassú”, döcögős volt, új funkciók viszonylag kis számban jelentek meg benne, ikonjai „csúnyák” voltak (talán sokkal inkább archaikusak).[6] Így amikor 2008-ban megjelent a kinézetében és funkcióiban számos újdonságot hozó 3.x.x-s sorozat, szinte mindenki a szokásos, régi szöveget mantrázta, s csak keveseknek tűnt fel, hogy valódi, nagy változások zajlottak a felszín alatt. Ez a változat volt az, amely már igazi, „minőségi” ugrást jelentett. Ehhez a sorozathoz készült el a Németh László által fejlesztett Lightproof nyelvhelyesség-ellenőrző (azóta megjelent angol és orosz nyelvekhez is), valamint a 3.2-s változatban bemutatkozó Graphite motorhoz[7] a valódi betűfokozatokat ajánló (lehetővé tevő), nyílt és szabad Linux Libertine betűkészlet módosított, s egyben nyelvspecifikussá is tett – szintén Németh László által fejlesztett – Linux Libertine G.[8] Az OpenOffice.org 3.3 bétájának forkjából készült LibreOffice fejlesztése pedig már egy másik történet lenne…

Valószínűleg 2010. március 1-ig kevesen gondolták, hogy lehetséges a kiadványkészítés az OpenOffice.org alkalmazással. Ekkor jelent meg Németh László jegyzete[9], mely gyakorlati példákat mutatott a legalapvetőbb szerkesztési ismeretektől egészen a mélyebb, profibb megvalósításokig.

A Neveléstudomány című folyóirat szerkesztésekor mi magunk[10] is ezen jegyzet későbbi változataiból[11] merítettünk. Természetesen a kísérletezés, a „játék” ezúttal sem maradt el: rengeteg próbálkozás, oldalrendezés, stílusállítás útján készültünk el a véglegesnek tekintett változattal, amely természetesen nem statikus.

Az alábbi kis írás ezen folyamatba kíván betekintést nyújtani, gyakorlati példák és problémák fel-, illetve bemutatásával, amelyekkel az eddigi munka során szembesültünk. Jelen szövegben nem kívánom megismételni a Németh László-féle jegyzetben szereplő megoldásokat, legfeljebb itt-ott egyéb megoldást mutatok – kissé talán bölcsészesen terjengősen.

[6] Azt kevesen tudták, hogy a helyesírás-ellenőrzője régóta sokkal jobb volt, mint a piacon kapható versenytársaké, a helyesírási szótára pedig – amely a firefoxhoz és a Google Chrome-hoz is használható – rendszeresen frissül a legnagyobb kereskedelmi versenytársával ellentétben.

[7] Az OpenOffice.org 3.2-ben megjelent újdonságokról részletes áttekintés olvasható az openoffice.hu weboldalán: http://openoffice.hu/2010/02/15/az-openoffice-32-ujdonsagai/

[8] A pontos történethez lásd a http://www.libreoffice.hu/ oldalon a „Libertine” vagy „betű” címkével kiemelt cikkeket.

[9] http://libreoffice.hu/2010/03/01/kiadvanykeszites-openoffice-org-writerrel-elozetes/ – A hír eredetileg az openoffice.hu weboldalon jelent meg.

[10] Schnellbach Mátéval (http://schnellbachmate.hu/) közösen tördeljük a folyóiratot.

[11] A jegyzet azóta *Kiadványszerkesztés LibreOffice Writer szövegszerkesztővel* címmel érhető el a következő webcímen: http://numbertext.org/libreoffice/

2. Kezdeti problémák és nehézségek

Amikor a folyóirat szerkesztésén – konkrét megvalósításán – elkezdetünk dolgozni, volt néhány fontos dolog, amit szem előtt kellett tartanunk, mely nem pusztán a magyar felsőoktatás „pénztelenségét” mutatja. Pontosabban fogalmazva: nem csak a pénz volt az elsődleges szempont, amikor a tördelés módján elkezdtünk dolgozni.

Mivel a folyóiratot nem „tartja el” semmilyen bevétel, alapítvány vagy szervezet, és a benne dolgozók önkéntesen vállalták a munkát (a kezdeti lépések megtételéhez a TÁMOP nyújtott segítséget), így ennek megfelelően kellett a szoftvereket, betűket is kiválasztanunk. Ez azt jelentette, hogy szabad szoftverek és szabad betűk jöhettek szóba, melyek lehetővé teszik a folyóirat publikálását az interneten anélkül, hogy szerzői jogot sértenénk.

Az online, tudományos folyóiratok egyik fő gondja, hogy hogyan képesek „komolynak, nagynak” látszani – kivívni maguknak státust, rangot – egy olyan világban, ahol az online – függetlenül attól, hogy egyre javuló minőséget produkál – alapvetően még mindig nem érte el a megfelelő státust a tudományos kutatói világban. Igaz ennek problémái meglehetősen sokrétűek, régiónként eltérőek (kutatóközösségi konzervativizmus, az online tartalmak megbízhatatlansága, a webfelületekkel való sorozatos visszaélések, a hivatkozások relatív volta stb.), ám az olcsóbbá nem váló papír alapú lapkiadás, és a megnövekedett igény a gyors és kevésbé korlátozott (színes képek, terjedelem stb.) szövegek kiadására egyfajta elmozdulást eredményez(ett) az online tartalmak felé.[12] Ezeket a folyamatokat a szabad szoftverek is elősegítik.[13]

Az online tartalmaknál – különösen a tudományos folyóiratok esetében – rendkívül fontos és kényes kérdés a hivatkozás. A folyóiratokon belül ez általában lábés/vagy végjegyzetek segítségével valósul meg – technikailag. Az online folyóiratokra való hivatkozás már korántsem ilyen egyszerű dolog.[14] Ebben szerepet játszik a papír alapú hivatkozási hagyomány digitális továbbélése (annak szándéka), ami az online folyóiratok esetében korlátozottan megoldható. A régi séma (szerző, cím, folyóirat és adatai) ma már elsősorban linkekben ölt testet, ám ezek gyakran elavulnak, a honlapokról pedig eltűnnek a korábbi tartalmak, aminek következtében a források „elérhetetlenné” (vagy legalábbis nehezen elérhetővé) válnak. Az online folyóiratok PDF vál-

[12] A „publikálj vagy pusztulj” (publish or perish) elv Magyarországon nem kizárólagosan érvényesül, de egy-egy felsőoktatási intézmény esetében, annak akkreditációjánál figyelembe veszik a publikációk számát és az oktatói, hallgatói (tudományos) tevékenységet is. A bölcsész- és társadalomtudományi területen a publikáció (valamint a hivatkozások és a diákköri versenyek) az egyik ilyen teljesítménybeli mutató.

[13] A mai szabad szoftveres technológiák (például a tartalomkezelő rendszerek (CMS), betűtípusok, webfelületen használható betűk (WOFF) stb. révén) nemcsak (platform)függetlenséget, hanem „időtállóságot” is jelentenek, mert a nyílt szabványok segítségével bármikor reprodukálhatóvá válnak a fájlok megnyitásához és szerkesztéséhez szükséges eszközök. Továbbá a nyílt szabványok garantálják a gyártófüggetlenséget, így nem fordulhat elő, hogy egy-egy fájlunk évek múlva nem, vagy csak igen nehézkesen lesz megnyitható. Ennek teszteléséhez nyissunk meg ma használt kereskedelmi, zárt szövegszerkesztőnkkel egy 10-15 évvel ezelőtt, ugyanezen szövegszerkesztő korábbi változatával készített dokumentumunkat.

[14] Az állami támogatással létrejött nemzeti archívumok - mint a Magyar Elektronikus Könyvtár (MEK) és az Elektronikus Periodika Archívum és Adatbázis (EPA) - valamennyit segítenek a hivatkozások rendezésében, de a kérdést megnyugtatóan még nem sikerült rendezni.

tozatai hivatottak fenntartani ezt a hagyományoshoz nagyon hasonló hivatkozási formát, s ezzel együtt mintegy megtartva a hagyományból fakadó tekintélyt.[15]

Ezeket a problémákat figyelembe véve kellett megtervezni a folyóirat kinézetét és a szerkesztőbizottságnak megmutatni.

3. Technikai nehézségek

Ahogy már fentebb jeleztem, csakis szabad szoftverek jöhettek szóba a folyóirat végső kialakításának elkészítéséhez. A betű kiválasztása sarkalatos pont volt a folyóirat életében. A diszlexiások számára könnyebben olvasható és a honlapok által is preferált talpatlan betűt (sans-serif) kellett találni, mely „minőségében" (esztétikusságában) is megfelelő,[16] valamint szabad licenccel rendelkezik. Több betűtípus is szóba került, úgy mint a Droid Sans, a DejaVu Sans, Free Sans, ám végül a KDE-ben már bevált Oxygen font mellett döntöttünk. Az Oxygen adja az ún. kenyérszöveget (szövegtörzset), a címsorokhoz pedig a magyar fejlesztésű, „Linux Libertine G"-t választottuk.[17] Így jól elkülönülnek a címsorok a kenyérszövegektől. Mindkét betűvariáns tartalmazza az f-alapú unicode-os ligatúrákat, melyek olvashatóbbá teszik a folyóiratot.

A programválasztásnál az egyedüli megoldásnak a mindkettőnk által ismert, használt LibreOffice jöhetett szóba. Mivel Microsoft Windowson és Linuxon is megtalálható, valamint a windowsos változatból hordozható (portable) változat is rendelkezésre áll, így akár az egyetemen, akár otthon, laptopon is tördelhetjük a lapot. A LibreOffice melletti további érv volt számunkra, hogy folyamatosan javul a Microsoft Office zárt formátumaival (doc, docx) való kompatibilitása,[18] ezenfelül a beépített, automatikus mechanizmusok (stílusok szervezhetősége, szövegek tömbösítése) jelentősen meggyorsítják a lap tördelését.

A LibreOffice melletti további érv annak támogatottsága. A fejlesztőkhöz és a fórumokhoz mindig bizalommal fordulhattunk, ha bármilyen kérdésünk volt, vagy csak elakadtunk a megvalósítás rögös útján. A LibreOffice-hoz készített – elsősorban magyar fejlesztésű – kiegészítő, a Lightproof (ligatúrák, ezrestagolás, idézőjelek stb. figyelmeztetője), nagy segítségünkre van a tördelésnél.

Mivel a szerzők az esetek túlnyomó részében nem használják ki a szövegszerkesztők beépített szövegstílusait, így a formázás elmarad (néhány szövegbeli kiemeléstől eltekintve, mint például a dőlt betűk használata). A tördelés legegyszerűbben a Ctrl+C és Ctrl+V billentyűkombinációk alkalmazásával történik, egy előre elkészített és megformázott ODT-állományba beillesztve a kivágott szöveget.[19]

Ahhoz, hogy a lapra offline módon is hivatkozni lehessen, olyan kimeneti formátumra volt szükség, mely garantálja a platformfüggetlenséget, vagyis minden rendszeren, legyen az táblagép, laptop, személyi számítógép, vagy Macintosh, ugyanazt kapja az olvasó. Így egyetlen elterjedt, mindenütt támogatott, hordozható (lementhető) ki-

[15] Számtalan online folyóirat él azzal a módszerrel, hogy PDF-jében leírja a hivatkozás formáját.

[16] A diszlexiások részére külön kifejlesztett, szabad betűtípus is elérhető a http://dyslexicfonts.com/ címen.

[17] A címsorok jó elkülönítéséhez nem pusztán egy eltérő – talpas – betűt használtunk, hanem szaggatott vízszintes vonalat is.

[18] A szerzők többsége ugyanis a Microsoft Office valamely változatát használja a szövegei elkészítéséhez, melyet végül annak zárt formátumában ment.

[19] Próbálkoztunk a korrektúrázott, olvasó-szerkesztett szövegek közvetlen formázásával is, ám az felesleges plusz munkákat és „szöveggondozást" igényelt volna.

meneti formátum jöhetett szóba: a PDF. A LibreOffice képes PDF formátumba menteni, abba becsomagolni az eredeti ODT-állományt is (hibrid PDF) ezzel a későbbi szerkeszthetőség sem vész el.

4. Az online lapszerkesztés sajátosságai

Az online folyóirattal kapcsolatban, a tudós lélek első gondolata, hogy terjedelmi korlátok nélkül készítheti el – vagy éppen írhatja ki magából – tanulmányát, régóta érlelt gondolatait. Természetesen nem így van, hiszen az online olvasók sem a végtelen befogadására vállalkoznak, sőt éppen ellenkezőleg.[20]

A Neveléstudomány esetében egyáltalán nem változtattunk az oldalméretezésen, vagyis meghagytuk a sztenderd A4-es oldalt. Erre nem saját, kényelmi okokból volt szükség, hanem azért, mert a nyomtatások túlnyomó többsége A4-es oldalra történik, így célszerűnek láttuk megmaradni ennél a formánál, és nem az A5 valamely változatát alkalmazni. Ennek további előnye, hogy ha valaki mégis a lap nyomtatásáról dönt, akkor ugyanazt fogja kapni offline (kinyomtatva), mint online, s az általa preferált formában olvashatja el. Mivel nem kerül sor könyvszerű – offline – kiadásra, így a belső margóknál a kötés helyét nem kell figyelembe venni.[21]

A címsorok mindig is sarkalatos pontjai voltak a szövegeknek, monográfiáknak. Egyrészt ezek azok az igazodási pontok, melyeket felhasználva generálódik a tartalomjegyzék (erről még később lesz szó), másrészt a folyamatos szöveget ezek segítségével lehet jól elkülöníthető részekre tagolni, melyek egyben a szöveg struktúráját is jelentik. Egy folyóirat, vagy szerkesztett kötet esetében nem csak az egyes tanulmányok, hanem az egész kötet struktúráját is ezzel alakítjuk ki.[22] A címsorok kialakítása a Neveléstudomány esetében is a LibreOffice alapértelmezett címsorainak módosításával történt, a Címsor 4-től (egy-egy tanulmány címsorai) beiktatva egy vízszintes szaggatott vonalat. Szokás a címsorok számozása – főleg a tanulmányokon belül – ám mi úgy ítéltük meg, hogy bár könnyen megoldható a LibreOffice automatizmusával, felesleges volna a címsorok „elcsúfítása" számok beiktatásával.

A talpatlan betűk mérete – mivel többségében a címsorok megformázására használják őket – általában 12 pt esetében is nagyobbak, mint a talpas betűk mérete (s ez az Oxygen betűtípus esetében sem volt másként), ezért mi a kenyérszövegben 10 pontra módosítottuk. Ennek következtében meglehetősen sűrű – „szirup sűrűségű" – blokkokat (szövegtörzset) kaptunk, melyeket tovább sűrített a bekapcsolt, automatikus elválasztás. A szellősséget a sorközök növelésével – egyszeresről, rögzítettre (0,55 cm értéket megadva) – sikerült elérni. Így egy jól strukturált, olvasható szöveget kaptunk. A szerzők gyakran alkalmaznak a szövegen belül számozott felsorolásokat. Ezek esetében mi úgy döntöttünk, hogy a folyóiratban jobbra igazítva formázzuk meg, ezáltal pontosan egymás alatt jelennek meg a számok és a szövegek – könnyítve az áttekintést.[23]

[20] Felesleges emlegetni azokat a kutatásokat, amelyek éppen azt bizonyítják, hogy a hosszabb, képeket egyáltalán nem, vagy alig tartalmazó cikkeket kevesen olvassák.

[21] Ez a gyakorlatban az jelenti, hogy tükrözött oldalelrendezés esetén a belső, vagy bal margóknál még körülbelül fél centit kellene számolni.

[22] A generálható tartalomjegyzék elengedhetetlen, ha nem akarjuk a bizonytalanabb és sok esetben (tipográfiailag is) pontatlanabb kézi megoldással elkészíteni a tartalomjegyzéket. Egy jól áttekinthető, kellően strukturált és szellős tartalomjegyzék az olvasót is segíti.

[23] A szakirodalom felsorolásánál is ugyanezt a jobbra-igazított felsorolást alkalmazzuk.

Ilyen terjedelmű (mennyiségű) szöveg megformázásához elengedhetetlenek az oldalstílusok. Ezekkel a különféle rovatok „címoldalait” (például Fókusz, Körkép, Nézőpontok, Kulcskérdés) formázzuk meg.[24]

Az egyedi kinézet kialakításának fontos eleme a jobb alsó sarokban, a szövegszéleken kívül elhelyezett, nagyméretű oldalszám, továbbá a lábjegyzetek esetében alkalmazott szaggatott, vízszintes vonal (lénia). További egyediséget biztosít a tanulmányok rövid összefoglalására szolgáló absztrakt, melyet mi a Németh-féle jegyzet 31. oldalán található megoldás adaptálásával oldottunk meg.[25]

A címsoroknál már említettem a tartalomjegyzék kérdését. Mivel a tartalomjegyzék nem csak jelenlétével, hanem kinézetével is informálja az olvasót, fontos, hogy gyorsan áttekinthető, a tudományos folyóiratoknál már megszokott formátumot kövesse, attól ne nagyon térjen el. A LibreOffice esetében ez is lehetséges, igaz ennek megoldásához már „külső segítséget” kellett igénybe vennünk. Németh László esettanulmányát[26] könnyen átültettük a gyakorlatba, vagyis a folyóirat tartalomjegyzékének elkészítését is sikerült automatizálni, ezáltal a Neveléstudomány tartalomjegyzéke generálódik (nem kell kézzel „kiszedni”) és megfelel a megszokott, tudományos elvárásoknak is.

5. Összefoglalás

Ugyan a folyóirat technikai szerkesztése hallatlanul izgalmas és érdekes feladat, mindezek a munkák nem jöhetnének létre, ha nem lenne egy profi és lelkes szerkesztőbizottság, amelynek tagjai a folyóirathoz érkező szövegeket elődolgozzák (kiküldik anonim bírálatra, olvasószerkesztik, korrektúrázzák stb.) és rengeteg apró, szöszölős munkát elvégeznek, mire az asztalunkra (technikai szerkesztőkére) kerül, hogy elvégezzük rajtuk a végső simításokat.

A Neveléstudomány bár rögös úton születik, változatos platformokon, mégsem jöhetett volna létre, ha nincs a szabad szoftvereket fejlesztő közösség (a hol mögöttük álló, hol mögülük kihátráló cégekkel), akik létrehozták azt az ökoszisztémát, amelynek felhasználásával elkészül a folyóirat (és elkészült ez a kis írás is).

A Neveléstudomány megjelenő számai visszaigazolják nemcsak a készítők szándékait, hanem a LibreOffice és a szabad szoftverek fejlettségét és használhatóságát, flexibilitását, bizonyítva, hogy az ingyenesség nem jelent mindig mindenben alkut a minőség rovására.

A Neveléstudomány honlapja: http://nevelestudomany.elte.hu/

[24] Még korai ötlet gyanánt felmerült a kéthasábos tördelés a recenziók, ismertetések esetében – oldalstílusok felhasználásával –, ám ezt elvetettük.

[25] Terveink között szerepel, hogy az évi négy lapszámban eltérő színekben pompázzon az absztrakt, ezáltal is könnyítve a folyóiratszámok megkülönböztetését.

[26] Az esettanulmány elolvasható a http://libreoffice.hu/2012/07/03/esettanulmany-folyoiratok-tartalomjegyzeke/ címen.

KIADVÁNYSZERKESZTÉS LINUXON, AVAGY A SCRIBUS NYOMDÁSZ SZEMMEL…

Pércsy Kornél

Kivonat

Pár éve foglalkozom – elsősorban interneten megjelenő – magazinok szerkesztésével. Többek között a JAF Magazin, Full Circle Magazin magyar kiadásának kiadványszerkesztésével. Mivel a JAF Magazin egy hobbiból készülő magazin, ezért nem állt szándékomban több százezer forintért beszerezni kiadványszerkesztő programot. Kerestem a megoldást, hogyan lehetne olcsón megúszni egy ilyen feladat kivitelezését. Nem sok sikerrel jártam, mivel kevés ingyenes, de jól használható programot találtam. A választás a Scribus programra esett, mivel a Full Circle Magazin eredetije is ezzel a programmal készült, és ezt kellett használni a magyar kiadás szerkesztéséhez is. Azt mondhatom, emiatt kezdem eléggé kiismerni magam ebben a kiadványszerkesztő programban. Úgy gondoltam, összeállítok egy ismertetőt, és megosztom a tapasztalataimat, remélve, hogy ezzel tudok segíteni másoknak is.

Tartalomjegyzék

1. Egy kis történelem

A történethez hozzátartozik, hogy 1980 óta nyomdában dolgozom. Van három nyomdaipari szakmám, amelyek közül az egyik már „kihalt". Nem azért, mert olyan öreg lennék, hanem mert az elmúlt időben nagyon sokat változott a nyomdaipar a technikai fejlődések miatt.

A nyomdai történelemről csak vázlatosan egypár szót mondanék. Eleinte, ha valamilyen szöveget ki akartak nyomtatni, külön-külön betűkből kellett összerakni. Ezen az ábrán ilyen betűket láthatunk.

14. ábra

Egy kicsit modernebb technika, amikor egy ilyen gépen kellett a szöveget „legépelni," és egy sornyi szöveget öntött ki a gép ólomból.

Az ofszet technológia kezdetén a kézzel, illetve géppel szedett szöveget kinyomtatták egy példányban, és ilyen fényképezőgéppel lefényképezték, majd azt használták tovább.

15. ábra

A technika további fejlődése során már nem ólombetűket használtak, hanem „fényszedő” géppel egyből filmre világították a nyomtatni valót. Később forgalomba jöttek a szövegszerkesztő, tördelő programok, amivel el is jutottunk a ma használt technikákhoz.

16. ábra

Összegyűjtöttem egypár kiadványszerkesztő program árát, amit manapság a leggyakrabban használnak.

Adobe FrameMaker	380 000 Ft
Adobe InDesign	260 000 Ft
Adobe PageMaker	0 Ft
Apple Pages	0 Ft
Corel Ventura	350 000 Ft
Microsoft Publisher	50 000 Ft
QuarkXPress	350 000 Ft
Passepartout	0 Ft
Scribus	0 Ft
TeX, LaTeX, MaTeX, LyX	0 Ft
Troff	0 Ft

2. A Scribus program áttekintése

Az előzőek alapján – és amint a bevezetőben említettem – a Scribus programot tekintjük át. Nem részletes programbemutatót akarok tartani, hanem inkább rövid általános bemutatót, és a véleményemet, tapasztalataimat a programról.

17. ábra

Indítás után ez a képernyő fogad bennünket. Itt van lehetőségünk kiválasztani, hogy új dokumentumot készítünk, vagy megnyitunk egy már elkezdett dokumentumot, esetleg a legutolsó szerkesztett dokumentumok közül valamelyiket. Az új dokumentumnak itt állíthatjuk be a tulajdonságait, méreteit – margók, kifutók, stb.

Ebben a programban is megtaláljuk a szokásos menüket, illetve néhány extra me-

18. ábra

nüt, amelyek a kiadványszerkesztéshez szükséges funkciókat teszik elérhetővé. Ezeket most nem fogom részletezni, mert már az is meghaladná egy előadás kereteit. Csak futó pillantást vetünk rájuk: Fájl menü: új, megnyitás, mentés, export, import, nyomtatás, kilépés, stb. A Szerkesztés menüben is az ismert menüpontok találhatók: kijelölés, másolás, kivágás, beillesztés, stb.

19. ábra

Következő: Elem. Minden olyan dolog amit az oldalon elhelyezünk. Ezek kezelésére, tulajdonságai beállítására szolgáló menük.

Van egy külön „Beszúrás” menü, amely nem azonos a „beillesztéssel”. Mivel minden elem egy „keretben” van elhelyezve, ebben a menüben tudunk különféle típusú kereteket elhelyezni az oldalon. Ezekről a keretekről egy kicsit később.

20. ábra. Oldalak kezelése, nézet…

21. ábra. Extra lehetőségek, parancsfájlok, ablak – az ablakok elhelyezése, látható ablakok kiválasztása…

3. Amitől már a profi programok közé sorolható

Az előzőkben már láthattunk egypár olyan lehetőséget, amelyek révén a profi programok közé sorolható a Scribus. Lássuk konkrétabban, melyek is ezek.

Ezen a képen a Full Circle Magazin egy korábbi kiadásának egy oldala látható. Az oldalon találunk szöveget, iniciálét, képet, egy emblémát, és így elsőre nem látszik, de egy hivatkozást is. Mint említettem, minden, ami az oldalon van, az egy-egy keretben van elhelyezve. Szövegkeret, képkeret, egy téglalap a címsáv, és egy pdf megjegyzés.

22. ábra

Az oldalon lévő szöveg egy szövegkeretben van, amely 4 hasábosra van beállítva. Az oldal közepén van egy kép, amely nem takarja el a szöveget, hanem a szöveg körbeveszi a képet. A kép körül látható téglalap alakú keret a képkeret. Viszont a szöveg úgy helyezkedik el, hogy a kereten belül is látszik, de a képre nem lóg rá.

23. ábra

A magyarázat, hogy minden kerethez tartozik egy kontúrvonal. Lehetőség van rá hogy ne a keretet vegye körbe a szöveg, hanem a kontúrvonalat.

24. ábra

A tulajdonságok között állíthatjuk be a szöveg körbefuttatását. Itt látjuk, hogy tilthatjuk a körbefuttatást, vagy beállítható, hogy a keret alakja körül fusson a szöveg,

mert a keret lehet akár kör, háromszög alakú is. De akármilyen alakú is a keret, a határoló doboz mindig négyszög alakú, amelybe belefér az alakzat. Az utolsó lehetőség, amit ezen az oldalon alkalmaztunk is, a már említett kontúrvonal körüli szöveg futtatás.

25. ábra

Az előző képen a tulajdonságok közül az alakzat részt néztük meg. A tulajdonságok között találhatók továbbá a koordináták, a csoport, a szöveg, a kép, a vonal, és a szín.

A szöveg beállításai további részekre van bontva: betűtípus, méret, sortávolság, színek és hatások, stílusok, hasábok, optikai margók, és haladó beállítások: betűközök, betűtorzítások, stb. Ezek a beállítások jóval több lehetőséget adnak, mint egy szövegszerkesztő, vagy egy egyszerű kiadványszerkesztő program.

Egypár olyan lehetőséget szeretnék bemutatni, ami tényleg a profi programok közé emeli a Scribust. Az egyik például a gyűjtőalbum. Ez egy nagyon hasznos eszköz. Nyilván mindenki használt már vágólapot. Ott általában egy kimásolt, kivágott elemet tudunk tárolni, és újra felhasználni. Általában addig, amíg a gépet ki nem kapcsoljuk. Ezzel ellentétben a gyűjtőalbumba, mint a neve is utal rá, több elemet tudunk elhelyezni, ráadásul csoportokba rendezve. Akár egy egész oldalt – képet, szöveget – is küldhetünk a gyűjtőalbumba.

26. ábra

Ami pedig a lényeg: amíg az albumból nem töröljük az elemeket, addig bármikor felhasználhatjuk. Még a gép újraindítása után is.

27. ábra

Szintén a profi programok egyik tulajdonsága a fejlett keresés és csere funkció. Gyakorlatilag egy szöveg bármilyen tulajdonságú tartalmára kereshetünk, illetve bármilyen tulajdonságú tartalomra cserélhetjük.

28. ábra

Egy lehetőség a szöveg szerkesztéséhez, egy külön szövegszerkesztő ablak. Itt egy egyszerű betűtípust használva könnyen és gyorsan lehet a szöveget összeállítani. Az ablak bal oldalán a szöveg kijelölése nélkül állíthatjuk be a bekezdések stílusát. Ezenkívül a felső menüsorban bármilyen szövegbeállítással változtathatunk a beállított stíluson, például néhány szó kiemelését, anélkül hogy a bekezdés stílusát változtatnánk.

29. ábra

Ha már szóba kerültek a stílusok, itt egy ablak, ahol a stílusokat kezelhetjük, beállíthatunk vonal-, bekezdés- és karakterstílusokat. Itt tudunk beállítani például iniciálét, illetve egy stílust az iniciálés bekezdéshez.

Egy nagyon hasznos eszköz a képeffektusok lehetősége. Itt a képek megjelenítésén változtathatunk. Ezek a lehetőségek a kereskedelmi programoknál – tudomásom sze-

30. ábra

rint – csak külön bővítőmodulok vagy külső program használatával érhetők el. Ez is egy plusz pont a Scribus mellett.

31. ábra

Eddig még nem említettem, de a Scribusban is van beépített vektorgrafikus szerkesztő. Kellő türelemmel, gyakorlattal vektoros rajzokat is tudunk készíteni, szerkeszteni. Az egyszerűbb megoldás viszont betölteni egy vektoros rajzot. A már kész rajzolt, vagy betöltött vektoros grafikákon az útvonal eszközzel tudunk effektusokat alkalmazni. Például lencse effektust, nagyító, vagy halszemoptika szerinti torzítást, akár többet is egy objektumon. Egy másik effekt a hálós torzítás. Az objektumot betakaró háló metszéspontjait mozgatva tudjuk a rajzot torzítani. Ezek az effektek is a többi programnál általában csak külső alkalmazások használatával lehetségesek. Természetesen ezek a külső programok, vagy kiegészítő lehetőségek mind-mind – nem kevés – plusz pénzbe kerülnek. Még egy lehetőséget szeretnék bemutatni.

Nyomtatás előtt van lehetőség egy nyomtatási előnézet megtekintésére. Ez a nézet azért érdekes, és azért érdemel említést, mivel sok lehetőséget ad. Az átlátszóságok megjelenítése, élsimítás, és ami a lényeg: a színrebontás – színenkénti megtekintés. Ez a funkció is elég ritka a többi programnál.

Korántsem sikerült minden funkcióját bemutatni a Scribusnak. Remélem azonban, hogy felkeltettem az érdeklődést a program iránt, bizonyítva azt, hogy talán nemcsak az ára miatt érdemes használni, akár komolyabb nyomdai munkákhoz is. Természetesen sok javítani-fejleszteni való van még a Scribusban, de látva az eddigi fejlődését, bízom benne, hogy egy nagyon profivá váló kiadványszerkesztő programról beszéltünk. Egy utolsó biztatásként megemlítem, hogy egypár sikeres színes kiadványt készítettem már ezzel a programmal.

E-MAIL ARCHIVÁLÁS

Sütő János

Kivonat

Feltételezem, az olvasó menti a fájlszervereken, stb. lévő adatokat. Az is valószínű, hogy a levélkiszolgálón lévő postafiókokat is menti szalagra vagy diszkre. Azonban a hagyományos mentési eljárások nem nyújtanak megfelelő megoldást a vállalkozásokat érő kihívásokra. Ebben az írásban azt mutatom be, hogy miért érdemes archiválni a leveleket, és ismertetek egy nyílt forráskódú e-mail archiváló megoldást.

Tartalomjegyzék

1. Miért archiváljunk?

Több jó oka is lehet annak, amiért érdemes archiválni a leveleket. Az egyik legfontosabb az, hogy egy vállalat tudásának 70-80%-a a levelekben van. Az e-mail archiváló megoldások minden levélről egy másolatot őriznek, és így védenek az adatvesztés ellen.

Főleg Nyugat-Európában és Amerikában bizonyos iparági szereplők (pl. pénzügy, távközlés, egészségügy, államigazgatás, stb.) számára kötelező az elektronikus levelek meghatározott ideig (jellemzően évekig) történő megőrzése. Ha pedig egy hazai vállalkozás az USA-ban is végez üzleti tevékenységet, akkor – iparágtól függően – rá is érvényesek lesznek az FRCP, SOX, HIPAA, GLBA, SEC, NAD és társaik. Magyarországon még nincs kifejezetten az e-mailekre vonatkozó szabályozás, de várható, hogy a jövőben nálunk is megjelenik valamilyen, az e-mailekre vonatkozó regula. Az e-mail archiváló megoldások biztosítják a szabályozásoknak megfelelést.

Főleg az USA-ban „divat", hogy egyes cégek a fülemüle hangjáért is perre mennek egymással. Ilyenkor az a szokásos eljárás, hogy a felek jogászai a tárgyalás előtt kikérik egymástól az üggyel kapcsolatos dokumentumokat, különösen is az e-maileket, hogy aztán az ezekből nyert információkkal menjenek a bíróságra. Előbb azonban az informatikusoknak elő kell bányászniuk pl. szalagról (sok szalagról) az érintett leveleket. Ez azonban egy rendkívül munka- és időigényes folyamat, amely nagyon költséges. 2010-ben az USA-ban 80 milliárd dollárt emésztett fel. Egy e-mail archiváló megoldás lényegesen lerövidíti és olcsóbbá is teszi ezt a folyamatot.

32. ábra: Csökkenti a levelezőszerver terhelését

Ha egy kicsit technikaibb vizekre evezünk, akkor azért is jó ötlet a levelek archiválása, mert így csökkenthető a levelezőszerver terhelése. Több helyen kvótát alkalmaznak, nehogy a felhasználóknak sok GB-nyi levelük halmozódjon fel. Viszont a felhasználóknak az évek során több GB-nyi levelük is összegyűlik. Ezt az egymásnak feszülő

ellentétet a levelek archiválásával lehet feloldani úgy, hogy a szerveren pl. csak az utolsó 365 nap leveleit hagyjuk meg, míg az archiváló gépen az összes levél elérhető. Ez utóbbi azért is lehetséges, mert az archiváló megoldások a levelezőszervereknél hatékonyabban, illetve gazdaságosabban használják fel a rendelkezésre álló lemezterületet tömörítés, illetve deduplikáció segítségével.

Még a felhőben (pl. Google Apps) lévő postafiókokat is érdemes archiválni. Mert így akkor is hozzáférünk a leveleinkhez, ha a szolgáltató esetleg elérhetetlen, vagy ha 30 nap után jut eszünkbe, hogy mégis fontos volt a kitörölt levél.

A levelek mentése azért is nehéz feladat, mert a postafiókok tartalma folyamatosan változik (és nő), illetve sok levelet kell menteni. Egy e-mail archiváló megoldás úgy oldja meg a feladatot, hogy nem periodikusan akar nagyot markolni, hanem folyamatosan tárolja a másolatban kapott leveleket.

Periodikus mentések esetén azzal is számolni kell, ha a levelezőszerver 2 mentés között hibásodik meg, akkor az utolsó mentés óta átvett levelek elveszhetnek. Egy archiváló megoldás használata esetén nem fenyeget ez a veszély, mert azonnal megkapja, és tárolja a másolatot. Az adatvesztések oka egyébként leggyakrabban (30-60% közé teszik az interneten elérhető források) az ember.

2. Mentés vagy archiválás

Noha van értelme, és adott esetben jól is jöhet, ha a leveleket szalagra vagy diszkre írjuk, ez még nem e-mail archiválás. Az sem e-mail archiválás, ha a felhasználók PST fájlokat készítenek, és azokat a gépeiken tárolják – akkor sem, ha egy hálózati meghajtóra teszik a PST-ket. Végül az sem e-mail archiválás, ha pillanatfelvételeket készítünk a postafiókokról.

Mi hát akkor az e-mail archiválás? Az, ha a vállalati előírásoknak, illetve a vonatkozó jogszabályoknak megfelelően bizonyos ideig tárolunk bizonyos leveleket. Azt nyilván nem kell tárolni, ami nem hordoz értéket a cég számára, pl. levélszemetek.

3. A piler válasza

A piler egy nyílt forráskódú e-mail archiváló megoldás, amely egy központi, biztonságos archívumot képez, ahová a levelek automatikusan, emberi közreműködés nélkül kerülnek be, így biztosítja a vállalat üzletmenetének folytonosságát akkor is, ha a levelező szerver elérhetetlen. A piler hatékonyan használja ki a rendelkezésre álló tárterületet a tömörítés és a deduplikáció segítségével. A piler további erénye a gyors full text keresések lehetővé tétele.

4. Hogyan működik?

A piler lényegében egy SMTP szerver, amely SMTP protokoll segítségével fogadja, illetve dolgozza fel a leveleket. Az e-mail archiválás elkezdéséhez tehát csak annyi kell, hogy a vállalati levelezőszerver minden rajta áthaladó levélről küldjön egy másolatot az archiváló gép felé.

33. ábra: Piler a hálózaton

A megoldás része egy webes GUI is, amelyet a felhasználók autentikáció után érhetnek el. A piler démon MySQL adatbázisba írja az adatokat, a sphinxsearch pedig onnan veszi indexelésre a levélben található szavakat, címeket, stb. Amikor a felhasználók a webes felületen keresnek, akkor a GUI a MySQL adatbázisból, illetve a sphinx indexekből olvassa ki a szükséges adatokat.

5. Ásó, kapa és az internet se válasszon el

A szokásos felállásban a levelezőszerver és a piler a cég hálózatán belül helyezkednek el, de nem kell ennek feltétlen így lennie. Az is járható út, ha a levelezőszerver a cég hálózatán van, az archívum pedig a felhőben. Vagy fordítva, pl. ha a cég levelezését a Google Apps biztosítja, akkor a levelezőszerver van a felhőben, és az archívum lehet a cég hálózatán. Végül harmadik lehetőségként lehet a levelezőszerver az egyik felhőben, míg az archívum a másikban.

Ha a levelezőszervert és az archívumot hálózatok választják el egymástól, akkor jó, ha titkosítással védjük a leveleinket attól, hogy mások lehallgathassák vagy módosíthassák út közben. Erre való a STARTTLS nevű SMTP kiterjesztés, amit támogat a piler. Ehhez mindössze egy PEM fájlt kell készítenünk, amely tartalmazza a titkosító kulcsot, illetve a tanúsítványt.

Ebben a felállásban egyébként az is hasznos, ha korlátozzuk az archívumba levelet küldő gépeket. Ez lehet egy dedikált hardveres tűzfal a piler előtt, vagy Linux esetén egy sima iptables szabály is. De a piler támogatja a tcp_wrappers függvénykönyvtárat, amely lehetővé teszi, hogy a hosts.allow és hosts.deny fájlokkal szabályozzuk a piler démonhoz való hozzáférést.

6. Importálás

A levelek forrása alapvetően az SMTP. Azonban valószínű, hogy egy vállalatnál már rengeteg e-mail van, amiket jó lenne valahogy az archívumba tenni. A pilerimport segédprogram ismeri az mbox és a Maildir formátumokat, de le tudja tölteni a leveleket POP3, illetve IMAP protokollok segítségével, de ha a libpst csomag is telepítve van, akkor még a PST fájlokból is tud importálni.

7. Tudsz titkot tartani?

A piler a titkosított leveleket is archiválja, de azok tartalma nem kereshető. Ennek az az oka, hogy a levél dekódolásához, illetve indexeléséhez szükséges a címzett privát kulcsa, amelyet a pilernek odaadni még a cég dolgozói esetében sem szerencsés, a vállalaton kívüli levelezőpartnerek kulcsainak begyűjtése pedig képtelenség (legális úton). Viszont a titkosított levél (titkosítatlan) fejlécében szereplő adatokra lehet keresni, úgymint feladó, címzett, tárgy és dátum.

8. A levelek feldolgozása

Amikor a piler démon levelet kap, akkor azt feldarabolja, és kiveszi belőle, külön tárolja tőle a mellékleteket. Ha tehát egy levélben van egy PDF megrendelés és egy GIF logo, akkor az 3 fájlt eredményez: a 2 mellékletet, ill. a levél vázát, amelyben csak mutatók hivatkoznak a mellékletekre.

A piler minden levélhez egy 18 bájtos azonosítót rendel, amely 2 részből áll: egy TAI64 időbélyegből, illetve néhány véletlen számból a random device-ból. A levél egyes darabjai egy hash-elt, 3 szintű könyvtárszerkezetben tárolódnak, így biztosítva az egyenletes eloszlást az egyes könyvtárak között. A könyvtárak nevei a random bájtokból képződnek.

A jövőben valószínűleg módosul a könyvtárszerkezet úgy, hogy a legfelső szinten nem egy random bájt, hanem az időbélyeg egy része szerepel úgy, hogy kb. 12 nap levelei jussanak egy könyvtárba. Így jobban kezelhető egy esetleges mentés esetén. Azonban ez nem okoz törést a program működésében, mert visszafele kompatibilis lesz.

A verziók közötti upgrade-ek amennyire csak lehet sima, gond nélküli folyamatára külön hangsúlyt fektetek, ugyanis a saját leveleimet is a piler archiválja, és sokáig szeretném még azokat elérni.

Az egyes fájlokat zlib algoritmussal tömöríti a program, hogy minél kevesebb helyet foglaljanak el a diszken a levelek. A zlib ugyan nem a legjobban tömörítő algoritmus, de az egyik leggyorsabb kitömörítő.

Ezután jön a blowfish titkosítás, amely egy gyors titkosító algoritmus. A titkosításhoz egy a telepítéskor készített kulcsot használ a piler démon, amit az első levél után már nem szabad módosítani. Nagyon fontos, hogy ne veszítsük el ezt a kulcsot (különben nem érjük el az archívumban lévő leveleket), illetve hogy ne jusson illetéktelenek kezébe. A titkosítás egyébként a legtöbb (külföldi) szabályozásban (pl. FRCP, SEC, stb.) előírás.

A következő lépés a deduplikáció, amely során minden levélből, ill. mellékletből csak egyet tartunk meg. A deduplikáció 2 szinten működik: egyrészt a piler ellenőrzi a Message-ID fejlécet (amely kötelezően egyedi kell legyen), és ha egyezést talál, akkor eldobja a levelet naplózás után, hiszen az csak egy duplikátum. A piler minden mellékletre kiszámol egy SHA256 digest értéket, és megnézi, hogy szerepel-e az adatbázisában. Ha nem, akkor eltárolja a digest értéket és a mellékletet. Ha igen, akkor eldobja a mellékletet tartalmazó fájlt, és egy mutatót helyez el a már korábban tárolt mellékletre.

A levél ebben a fázisban archiválásra került, de a keresés még nem találja meg. A következő lépés az indexelés, amelyet a sphinxsearch nevű nyílt forráskódú termék

végez el. A sphinx kiolvassa a MySQL táblába írt, és sallangoktól megtisztított szöveget, majd egy indexet épít fel belőle, így a levél már kereshető lesz.

Ha nem tudjuk a keresett szó pontos alakját (pl. ragozott, tárgyesetben szerepel, stb.), akkor elég beírni a szó elejét, és egy csillagot tenni a végére. Így a sphinx az összes olyan levelet kidobja találatként, amelyikben szerepel az adott szótő. A szó elejének minimális hossza a sphinx.conf-ban van definiálva. A keresés eredményét egyébként cache-elhetjük Memcached démon segítségével, így a keresés eredményei közötti ide-oda lapozás gyorsabb lesz.

Időközben a piler kibővült a mellékletek indexelésének képességével is. A legutolsó stabil verzió már ismeri a text, ODF, doc(x), xls(x), ppt(x) és RTF formátumot.

9. Találj meg!

A felhasználók a webes GUI-n keresztül érik el a leveleket bejelentkezés után. A képernyő két részre van osztva. A felső részben lehet megadni a keresési feltételeket, illetve az alatt jelennek meg a találatok lapozható formában. A keresés eredménye rendezhető dátum, méret, tárgy és feladó alapján.

Esetleg azt gondolhatja az olvasó, hogy én is tudok keresni a Microsoft Outlookban vagy Thunderbirdben. Jogos. Azonban egy nagyobb méretű postafiókban való keresés meglehetősen megizzasztja a gépet. Jó, ha türelmesek vagyunk, és szeretünk gyönyörködni a homokórában, amíg az Outlook dolgozik. Ezzel ellentétben a sphinx sok felhasználó egyenként is szintén nagy postafiókjában is gyorsabban (és még az is lehet, hogy relevánsabb eredményt adva) keres, mivel ez egy erre a feladatra kihegyezett termék.

A komplikáltabb keresési feltételeket akár el is menthetjük, hogy ne kelljen újra begépelni. A keresés eredményéhez címkéket is rendelhetünk, amelyekben szintén kereshetünk.

A képernyő alsó felében a levelet nézhetjük meg, így ellenőrizve, hogy valóban ezt kerestük-e, illetve itt megjegyzést is fűzhetünk hozzá, amelyekben szintén kereshetünk. A keresés során megtalált leveleket nem csak megnézhetjük, de le is tölthetjük EML formátumban (több levél esetén ZIP fájlban), de akár SMTP-n egy kattintással vissza is állíthatjuk a postafiókunkba.

A levélváltásokat automatikusan felismeri a piler a Reference mező alapján, és a GUI-ban a levél mellett egy [+] jel mutatja, hogy az adott levél egy levélváltás része, és a jelre kattintva a teljes levelezési szálat láthatjuk.

A keresés eredményei mellett egy zöld pipa is van, ha a levelek integritás-ellenőrzését bekapcsoltuk. Ennek az az értelme, hogy meggyőződjünk arról, hogy az archívumból kiolvasott levél ugyanaz, mint amit eltároltunk. Ezt pedig a tárolt, ill. a kiolvasott levél digest értékeinek összehasonlításával állapíthatja meg a GUI. Eltérés esetén egy piros „x” jelenik meg a levél mellett.

10. Mindenki csak a magáét

A felhasználók csak a saját leveleiket érhetik el, amelyeket ők küldtek vagy kaptak. Ebből a szempontból az adminisztrátorok sem kivételek, ők annyival tudnak többet, hogy különféle statisztikákat nézhetnek meg a GUI-n, illetve adminisztrálhatják a piler beállításait. Az auditor jogú felhasználók viszont minden archivált levélhez hozzáférnek. Tipikusan a jogi vagy a HR osztály szokott ilyen jogosultságokat kapni egy nagyobb cégnél.

De honnan tudja a piler, hogy az adott levél kié? Ebből a szempontból az Exchange a legkellemesebb, amely a levélhez csatolt naplózó információkba beleírja a levél adatai között a levél feladóját és címzettjeit is. Második legjobb a Postfix, Kerio Connect és az Exim, amelyek az envelope címzetteket bele tudják írni a levél fejlécébe egy extra mezőbe, amit aztán a piler fel tud dolgozni. A többi SMTP szerver esetén a piler kénytelen a From, To és Cc mezők tartalmára hagyatkozni.

11. Szabály szabály hátán

A piler két szabályrendszert támogat. Az archiválási szabályok azt definiálják, hogy mit ne archiváljon a piler. Tipikusan a cég számára értéktelen leveleket érdemes kizárni az archívumból, pl. levélszemetek, a dolgozók vicces levelei, stb. A szabályokat a From, To, Subject mezők, továbbá a levél mérete alapján állíthatjuk össze a TRE regex könyvtár segítségével a GUI-ban.

A másik szabálycsoport a megtartási szabályok, amelyek azt határozzák meg, legalább meddig kell megőrizni az archivált leveleket. Ha nem adunk meg itt semmit, akkor alapértelmezetten 7 évre állítja a levél élettartamát. Ez nem jelenti azt, hogy a levél 7 év + 1 nap után törlődik az archívumból, hanem csak azt, hogy akkor már törölhető.

A törlést a pilerpurge program végzi el, amely akár le is tiltható a GUI-n. Ennek használata csak opció, lehetőség, nem pedig kényszer.

12. Valaki mentsen meg

Mivel az archiváló gép is csak egy számítógép, ezt is érdemes menteni egy másodlagos tárolóra. Erre alkalmas pl. a tar parancs, amivel pl. egy ~12 napos periódus leveleit foghatjuk könnyen össze, vagy a pilerexport program, amely az aktuális könyvtárba írja a megadott feltételeknek megfelelő leveleket, amelyeket aztán írhatunk szalagra, stb. Akár az is megoldható, hogy felcsatoljuk egy másik gép partícióját pl. NFS-en (vagy az iSCSI-n kiajánlott blokkeszközt), így a mentett levelek eleve egy másik gépre kerülnek.

De mi van akkor, ha az archiváló gép meghibásodik? A rendelkezésre-állás úgy is növelhető, ha a levelek és azok indexei nem az archiváló gépen vannak, hanem egy hibatűrő SAN/NAS rendszeren, amit az archiváló gép csak felcsatol. Ha pedig az archiváló gép valójában egy virtuális gép, akkor még rövidebb idő alatt klónozható belőle egy újabb példány, amely átveheti az előbbi feladatát. Amíg pedig az archiváló gép utóda el nem készül, addig a levelezőszerver tárolja a leveleket, majd ha elkészült az új archiváló gép, akkor továbbítja felé azt, ami addig összegyűlt. Így nem történik e-mail vesztés – hacsak nem pont akkor hal meg a levelezőszerver is…

13. Felhasználókezelés

A felhasználókat valahogy be kell mutatni a GUI-nak, hogy be tudjanak jelentkezni. Ennek a legegyszerűbb módja az, ha kézzel felvesszük őket a GUI-ban. Ez azonban 5 felhasználó után már nem túl vicces. Szerencsére van ennél jobb módszer is: a GUI képes LDAP protokollon letölteni a felhasználói adatbázist OpenLDAP vagy Active Directory szerverről. Cron-nal ütemezett feladatként beállítva a szinkronizációt automatikusan friss lesz a piler felhasználó adatbázisa.

A GUI támogatja a Google OAUTH megoldását is. Ebben az esetben a felhasználók a Google-nél autentikálják magukat, a jelszavuk nem is kerül a piler GUI-hoz, hanem sikeres autentikáció esetén (ha jóváhagyják, hogy a piler hozzáférjen a leveleikhez, akkor) a piler kap egy ún. hozzáférési tokent, amivel később is eléri a felhasználók leveleit,

Windows hálózatban biztosan szeretni fogják a Single Sign-on (SSO) funkciót, amelynek segítségével a felhasználók bejelentkezés nélkül használhatják a pilert. Valójában az történik, hogy a felhasználó böngészője (ami akár Firefox is lehet) és a piler GUI-ja mögötti webszerver, ill. a domain kontroller megbeszélik egymás között a háttérben az autentikációt.

14. 330 … 30 ezer

Mekkora vas kell a piler alá? Attól függ. Általánosságban elmondható, hogy memóriát használ a MySQL szerver, a sphinx komponensei, ill. az esetlegesen cache-elésre használt xcache, ill. Memcached is, továbbá a piler processzeknek is kell egyenként ~20 MB memória. Javaslom, hogy az archiváló gépbe legalább 2 GB memória kerüljön. Ha csak kipróbáljuk a pilert, akkor kevesebb is elég lehet.

A memóriánál komplikáltabb a szükséges diszk igény megbecsülése. Én személy szerint 1 TB alatt nem építenék archiváló gépet, hiszen évek levelezésének tárolásáról beszélünk. Fontos az is, hogy megfelelő I/O álljon rendelkezésre a piler, ill. a többi komponens számára. A későbbi bővíthetőség miatt mindenképpen LVM kialakítását javaslom, és természetesen valamilyen >= 1 RAID is melegen ajánlott a hibatűrés érdekében.

Ahogy korábban említettem, a /var/piler könyvtár lehet egy komolyabb, hibatűrő tárolón, míg az archiváló gép lehet egy blade-en vagy akár virtuális gépen is.

A lényeg az, hogy ne egy a kukából megmentett gépből csináljunk archiváló gépet, mert ha a felhasználók ráéreznek az archiválás ízére, akkor egyre masszívabban fogják használni ezt a szolgáltatást.

15. I want you to archive now

Remélem, hogy az előadás, ill. ez az összefoglaló meggyőzte az olvasót arról, hogy érdemes archiválni az e-maileket. A felhasználók imádni fogják érte. A főnöke is imádni fogja érte. Egy jó archiváló megoldás óriási segítség bármely cég számára, és sok terhet levesz az IT csoportról.

Eddig jobbára csak kereskedelmi e-mail archiváló megoldások voltak elérhetőek. A fehér holló számosságú nyílt forráskódú termékekből pedig több olyan funkció is hiányzik, amely meglehetősen kényelmetlenné teszi azokat a vállalatok számára.

Ezt az űrt szeretné betölteni a piler, amely a kereskedelmi megoldásokkal össze-

mérhető funkciókkal rendelkezik. Érdemes egy pillantást vetni a http://www.mailpiler.org/en/mailarchiva.html címre egy összehasonlító táblázatért.

16. Próbálj ki!

A http://www.mailpiler.org/en/demo.html címen elérhető egy demó, ahová néhány publikus levelezőlista levelei esnek be. Itt kipróbálható mindenféle szintű és jogosultságú hozzáférés. Aki a vállalatánál is szeretné egy pilot projektként bevezetni, tesztelni, de nem szeret telepíteni, az letölthet egy kulcsrakész VMware virtuális gépet is, amelyben csak az IP-címet, gép nevét kell beállítani és kezdődhet az archiválás. Ígérem, nem bánja meg, aki kipróbálja.

A piler felhasználók rengeteg ötletet, fejlesztési igényt vetettek fel a BitBucketen található projektoldalon. Ha az olvasó hiányol valamilyen funkciót, vagy hibát talált, akkor javaslom, hogy nyisson egy új hibajegyet a https://bitbucket.org/jsuto/piler/issues címen.

17. Köszönet

Végezetül itt is szeretném megragadni az alkalmat, hogy megköszönjem az FSF.hu Alapítványnak, hogy tavaly ilyenkor támogatásra méltónak ítélte a projekt ötletét. Továbbá Németh Ádámnak is köszönettel tartozom, amiért rengeteg építő kritikával segítette, hogy a webes GUI a korábbi „fapados, NB III-as” szint után mára egy sokkal élhetőbb külsővel rendelkezik.

LIBREOFFICE

Tímár András

Kivonat

A 2011-es Szabad Szoftver Konferencián is tartottam egy előadást a LibreOffice-ról és a Document Foundationről. Idén, 2012-ben is felkértek egy kiemelt előadás megtartására, illetve ebből egy cikk írására a követőkiadványba. A témám most sem lehet más, mint a LibreOffice, hiszen az elmúlt két évben szinte kizárólag ezzel foglalkoztam. Szerencsére az elmúlt egy év olyan mozgalmas volt a LibreOffice életében, hogy téma bőven van, nem kell ugyanazt elismételni, amit egy évvel ezelőtt mondtam.

Tartalomjegyzék

1. A LibreOffice létjogosultsága

A LibreOffice egy hagyományos asztali alkalmazás, szövegszerkesztő, táblázatkezelő, bemutatókészítő, stb. Az elmúlt időszakban felerősödtek azok a hangok, amelyek szerint a hagyományos asztali alkalmazások ideje lejárt. A jövő a cloud, a felhő, az emberek ott fogják tartani adataikat, és ezeket a legkülönfélébb eszközökkel az interneten keresztül érik el.

Az alkalmazásfejlesztésnél divatba jöttek a böngészőn belül futó alkalmazások, mondván, böngésző mindenütt van. Azonban ha megfigyeljük, még manapság sem jellemző, hogy az előadók a felhőhöz csatlakozva vetítenék le előadásukat, inkább a saját laptop saját merevlemezéről vagy pendrive-ról indítják a bemutatót. Ez érthető, hiszen mi van, ha elmegy a WiFi, nem kap a gép IP-címet, a DNS hibája miatt nem oldódik fel a távoli gép neve stb. Kellemetlenség adódhat az előre nem kiszámítható technikai problémák miatt. A web, mint platform sem megoldás mindenre, hajlamos vagyok múló divatnak tekinteni.

A webes platform két alappillérét, a HTML-t és a JavaScriptet eredetileg nem alkalmazásfejlesztésre tervezték, óriási erőfeszítések árán természetesen a feladat megoldható – elvileg. Gyakorlatilag a LibreOffice 8 millió kódsorból áll, 26 éve fejlesztik, és ha hihetünk az Ohloh számításának, 4046 emberévnyi munka van benne (COCOMO modell). Ebből az valószínűsíthető, hogy nem mostanában lesz kész az az alkalmazás, ami ugyanazt tudja, mint a LibreOffice, de webes platformon, JavaScriptben írva. A LibreOffice-hoz hasonló termékeknek még sokáig meglesz a piaca. A kormányzati szervek, az önkormányzatok, a vállalatok sokszor nem akarnak felhő alapú megoldást, féltik az adataikat. A LibreOffice a közoktatásban is sikeres lehet, szövegszerkesztést, táblázatkezelést, adatbázis-kezelést remekül lehet tanítani vele, de akár Logo programozás oktatására is alkalmas a LibreLogo eszköztár segítségével.

Nem kell, hogy mindenki LibreOffice-t használjon, elég annyi felhasználó, hogy a fejlesztői ökoszisztémát fenntartsa. A folyamatos, lépésenként haladó fejlesztés eredményeként azután sok minden meglesz az eddig nélkülözött funkciók közül is, például mobil eszközökön futó változat (Android, iOS), webböngészőben futó változat, közös dokumentumszerkesztés a hálózaton keresztül, kapcsolódás dokumentumkezelő rendszerekhez (pl. SharePoint) stb. Ezek a funkciók máris léteznek prototípus formájában.

2. The Document Foundation

A LibreOffice az OpenOffice.org-ból lett, az pedig a StarOffice-ból. Másfél évtized zárt forrású fejlesztést követően kb. egy évtizedig a Sun Microsystems, majd egy-másfél évig az Oracle határozta meg a program sorsát. Ez a felállás nem kedvezett sem az önkéntes hozzájárulóknak, sem a kódban érdekelt többi vállalatnak. A The Document Foundation (TDF) azért jött létre 2010-ben, hogy ezt a problémát megoldja, megszüntesse az egyetlen gyártótól való függőséget. Az alapítók az OpenOffice.org közösség vezetői közül kerültek ki, pontosabban azon közösségi vezetők határozták el az alapítvány létrehozását, akik nem voltak Oracle-alkalmazottak. A bejelentést követő hetekben az OpenOffice.org közösségből szinte mindenki, aki nem az Oracle-nél dolgozott, bejelentette, hogy csatlakozni kíván a TDF-hez. A TDF alapítói lefektették az azóta is érvényes alapelveket, tanulva a múlt hibáiból, más sikeres szervezeteket megfigyelve és megpróbálva megalkotni a tökéletes szabad szoftveres szervezetet.

A TDF független. Ez nem azt jelenti, hogy tagjai vagy vezetői között nincsenek je-

len a LibreOffice-ban érdekelt vállalatok alkalmazottai. Minden érdekelt fél jelen van, de a különböző vezető testületekben egyiknek sem lehet 1/3-nál nagyobb részesedése. A LibreOffice jövője nem egyetlen szponzor kezében van, a fejlesztőközösség sokszínű és kiegyensúlyozott.

A TDF meritokratikus. Ez azt jelenti, hogy annak van nagyobb szava, aki többet tesz le az asztalra. Ez logikus, hogy így van, összhangban van azzal az elvvel, hogy azok vezessék a projektet, akik fejlesztik. A TDF tagja az lehet, aki jelentős hozzájárulással segítette a LibreOffice-t. Ez lehet bármi, programozás, honosítás, dokumentációírás, marketing stb. A tagság elnyeréséhez legalább 3 hónapos „múltat" kell igazolni, egyúttal vállalni kell további 6 hónap aktív közreműködést. Egy vagy két ajánlóra is szükség van a tagsági kérelem elbírálásakor. A tagoknak választójoguk van és választhatók is a TDF vezető testületeibe. 2011 októberében zajlott le az első választás, a TDF-et irányító 7 tagú (+3 póttag) igazgatótanácsot választotta meg a tagság. Az igazgatótanács feladata a közösség építése, irányítása, a hivatalos (hatósági) ügyek intézése, az adománygyűjtés, a pénzügyek kezelése és a stratégiaalkotás. Azonban a LibreOffice fejlesztését illető kérdéseket eldöntő Engineering Steering Comittee tagjait nem a tagság választja, hanem a fejlesztők maguk közül jelölik ki, így nem fordulhat elő, hogy olyasmit szavaz meg a tagság, amit nem lehet megvalósítani vagy ellenkezik a projekten dolgozó fejlesztők elképzeléseivel. A tagnak jelentkezés folyamatos. Az elutasított emberek ismét jelentkezhetnek, ha jobban alá tudják támasztani jelentkezésük jogosságát.

A TDF alapításakor szándékosan nem rögzítették a szervezeti formát, hogy akik csatlakozni szerettek volna, szabadon alakíthassák ki a közös véleményüket erről. Az előkészületek kicsit több időt vettek igénybe a vártnál, de a cél egy nagyon stabil és megbízható szervezet létrehozása volt, ehhez sok erőfeszítés volt szükséges. A TDF ideiglenes vezetősége több lehetőséget is áttekintett, és végül az a döntés született, hogy az alapítvány szervezeti formája a német jog szerinti „Stiftung" legyen. A TDF németországi bejegyzéséhez minimum 50 000 euró volt szükséges. Ezért elindítottak egy nyilvános adománygyűjtő kampányt, hogy adományokból szedjék össze az 50 000 eurót. Körülbelül egy hónapot adtak a pénz összegyűjtésére. Sikertelenség esetén az Egyesült Királyságba vitték volna át a székhelyet, kicsit más jogi feltételek mellett. Az 50 000 euró azonban mindenki meglepetésére 8 nap alatt összejött, sőt az adományozók nem álltak le, körülbelül kétszer ennyi gyűlt össze az adománygyűjtési időszak végére. Megkezdődtek a tárgyalások az egyes német szövetségi államokkal, hogy kiderüljön, hogy melyikben lenne a legjobb helyen az alapítvány. A problémát azt jelentette, hogy a TDF elég speciális szervezeti formát mondhat magáénak. Egy klasszikus alapítvány élete úgy kezdődik, hogy egy gazdag ember vagy szervezet (az alapító) kitűz egy nemes célt, amelynek elérésére letesz egy bizonyos összeget, az alapítvány alaptőkéjét, és a vagyon kezelésére felkér erre általa alkalmasnak tartott embereket, a kuratórium tagjait. Innentől az alapító a pénz felhasználásába nem szól bele, és az alapítvány tevékenységének kedvezményezettje sem lehet. Az alapítványt kedvezmények sora illeti meg, adományokat gyűjthet, a nonprofit tevékenysége adómentes stb. A tagság fogalma ismeretlen ebben a kontextusban, az alapítványnak nincsenek (nem lehetnek) tagjai. A fentiekből látható, hogy TDF nem ilyen alapítvány. Az alapító összeget a közösség adta össze. Van tagsága, amely választja a kuratóriumot. A tagság elnyerése feltételekhez kötött, és nem örökös, hanem megújítandó. A TDF bejegyeztetését egy szabad szoftverek iránt elkötelezett német ügyvéd, Mike Schinagl intézte. Az

ő érdeme, hogy annak ellenére, hogy a TDF alapító okirata nem felel meg a német egyesülési törvényben lefektetett alapítványi definícióknak, hosszas egyeztetéseket követően elérte, hogy a berlini bíróság elfogadja azt, és alapítványként jegyezze be. A jog világában egy jogerőre emelkedett bírósági határozat érvényes akkor is, ha nem a törvény betűje szerint van, sikerült tehát egy olyan szervezetet létrehozni, amilyet terveztek, és amilyen még nem volt. Ez a jogi bravúr más német civil szervezetek érdeklődését is felkeltette, mintaként tekintenek erre az esetre.

3. Statisztika

Az OpenOffice.org fejlesztését az Oracle 2011 tavaszán beszüntette. Ezzel az OpenOffice.org projekt halottnak volt tekinthető. Az OpenOffice.org kódbázisára alapozva, a LibreOffice elindítása után pár hónappal elkezdődött az Apache OpenOffice fejlesztése is. A két projekt viszonyára inkább a versengés, mint az együttműködés jellemző. Ez teljes mélységében meg nem értett probléma, főleg ami az Apache OpenOffice mögött álló erők motivációját illeti. Mindez azonban kikényszeríti, hogy választ tudjunk adni bizonyos kérdésekre, amelyeket az újságírók és a felhasználók előszeretettel tesznek fel. Például hogy hány fejlesztője és hány felhasználója van a LibreOffice-nak.

Ezekre a kérdésekre nagyon nehéz pontos választ adni. Rendelkezésünkre állnak bizonyos adatforrások, például a verziókezelő rendszerünk (git) naplója, a wiki felhasználói adatbázisa, a fordítók által használt Pootle rendszer statisztikái, és a letöltési statisztikák a TDF által felügyelt rendszereken. Ezekből kellene kinyerni összesített adatokat, de belátható, hogy sok a bizonytalan tényező.

Nézzük először a „hány fejlesztő van” kérdést. A git naplójából azt tudjuk kinyerni, hogy pl. a projekt indulása óta hány egyén tett hozzá bármit is a forráskódhoz (vagy éppen vett el belőle, mert az is hozzájárulásnak számít). Csakhogy ennél a számnál sokkal érdekesebb lenne az, hogy jelen pillanatban hány fejlesztő van, de ezt ebből nem lehet megtudni. Beleszámoljuk viszont azokat az embereket is, akik nem is a LibreOffice-hoz szerettek volna hozzájárulni, hanem az Apache OpenOffice-hoz, csak a licenc által adott jogokkal élve a munkájuk átkerült a LibreOffice-ba. A kumulált hozzájárulószám azokat az embereket is tartalmazza, akik egyszer az életben küldtek valamit, aztán eltűntek örökre. Az összeg az csak egy szám, se az időfejlődést, se a jelen helyzetet nem írja le. Hasonló mondható el a wikiszerkesztőkről is. A fordítók közül azokat, akik online fordítanak a Pootle-ben, könnyű nyilvántartani. Érdekes egyébként, hogy az ott regisztrált 1131 felhasználó közül csak 330 az aktív, aki valaha fordított is valamit (akár csak egy szót). A többiek vajon miért regisztráltak? Azonban az offline fordítókról semmilyen adatot nem ismerünk. Tudjuk, hogy jó néhány fordítócsapat a Pootle-t csak a fájlok tárolására használja, más rendszerben vagy asztali alkalmazással fordítanak, és a csapat vezetője a „proxy” a fordítók és a Pootle között.

A rendelkezésre álló letöltési adatokból felhasználószámot gyártani igencsak merész dolog, de muszáj. A jelenleg érvényesnek tekintett 60 milliós becslés egyrészt a desktop Linux felhasználók számából (30 millió), a letöltési adatokból (20 millió), és az újság CD-mellékletek és vállalati felhasználók becsült számából adódik össze. Rengeteg a bizonytalanság. Már a letöltéseket sem lehet jól számolni, ugyanis nem egy kiadás volt, hanem sok. Az egyik véglet, hogy valaki mindet letöltötte, a másik, hogy valaki csak egy verziót töltött le. A nem TDF által ellenőrzött szerverek forgalmát még becsülni sem lehet. Valamennyire használható adat a központi frissítéskereső naplója,

de ez sem ad teljes képet, ugyanis vállalati környezetben az automatikus frissítés tiltott, Linux alatt pedig általában a disztribúció csomagkezelője végzi a frissítést.

4. Friss erők bevonása a fejlesztésbe

A LibreOffice projektnek a kezdetektől célja a növekedés, minél több új ember bevonása a fejlesztésbe. Az új fejlesztők mentorálása a régi fejlesztőkre ró terhet, de a gyakorlat azt bizonyítja, hogy a befektetett munka megtérül. Évente két konferencián, a brüsszeli FOSDEM-en és a LibreOffice konferencián (2011 Párizs, 2012 Berlin), valamint a tavaszi hamburgi és az őszi müncheni Hackfesten személyesen is találkozhatnak egymással a fejlesztők. Ezek az alkalmak mindig nagyon hasznosak, sokat tanulunk egymástól. Az IRC-n a freenode #libreoffice-dev csatornáján folyamatosan jelen vannak a fejlesztők, segítenek a kezdőknek. Magyarországon lehetőség van arra is, hogy egyetemi hallgatók kötelező szakmai gyakorlatukat a Novell magyarországi irodájánál töltsék el LibreOffice-fejlesztéssel. Ebben a szakmai gyakorlat programban az elmúlt másfél évben 15 hallgató vett részt, többségük sikerrel (aki akarta, megszerezte az aláírást).

A fejlesztők külön figyelnek arra, hogy mindig legyenek kezdők által megtámadható problémák, ezeknek az Easy Hack nevet adták. A projekt Easy Hack wikioldala állandóan frissül a megfelelő kulcsszavakkal ellátott Bugzilla-bejegyzések alapján. A feladatok nehézség és téma szerint is csoportosítva vannak, mindegyikhez tartozik mentor, és segítség az induláshoz, pl. hogy hol kell belenyúlni a forráskódba. A közösség segítségével így vált lehetővé az, hogy a LibreOffice forráskódjában elszórva elhelyezkedő mintegy 5000 nem használt metódus ki lett törölve, illetve a német nyelvű megjegyzések száma két év alatt a felére csökkent. Továbbra is vannak olyan feladatok, amelyek nem automatizálhatók, de egyszerűek, a kódbázissal (vagy akár a programozással) most ismerkedők is meg tudják oldani őket. Ilyenek például az elavult karakterláncosztályokról az Ostring, illetve OUString osztályokra áttérés, az RTL_CONST-ASCII_USTRINGPARAM makrók irtása, vagy a fix koordinátákat használó párbeszédpanelek átírása Glade XML formátumba. Az elképzelés az, hogy néhány ilyen egyszerű, bevezető feladat elvégzése után a kezdő fejlesztő megszerzi azt a rutint, ami a komolyabb, érdekesebb feladatok elvégzéséhez kell. De ha valaki soha nem lép tovább, akkor is nagyon hasznos szolgálatot tett a közösségnek. Minden javítás, kódtisztítás a többi fejlesztőt tehermentesíti, és a jövőben érkező fejlesztők dolgát teszi egyszerűbbé.

5. A LibreOffice 4.0 újdonságai

2013 februárjában fog megjelenni a LibreOffice következő verziója, a LibreOffice 4.0. A verziószám 4.0-ra változtatása a korábban várt 3.7 helyett nemcsak marketingszempontokkal indokolható. A 3-asról 4-esre váltó főverzió lehetővé teszi az eddig stabil, megváltoztathatatlan UNO API szükséges módosításait. A stabil UNO API elengedhetetlen volt a LibreOffice-hoz fejlesztett külső kiterjesztések helyes működéséhez, de egyes rosszul megtervezett vagy feleslegessé vált API-k karbantartása, toldozgatása egyre terhesebbé vált. Forradalmi változásra e téren nem kell számítani, az érintett kiterjesztésfejlesztőknek valószínűleg elég lesz a kiterjesztésüket egy új verziójú SDK-val újrafordítani.

Folytatódik az a trend, hogy a LibreOffice-t minél jobban együttműködővé tegyék más, zárt forráskódú szoftverek fájlformátumaival. Így a felhasználók meg tudják

nyitni ezeket a fájlokat olyan operációs rendszereken is, amelyekre az ezen fájlokat előállító szoftver nem érhető el, és tudják menteni a nyílt, szabványos OpenDocument formátumba. Egyes fájlformátumok leírása hozzáférhető, másokat vissza kellett fejteni. A LibreOffice 3.5-ben bemutatkozó Visio importszűrő a 4.0-ra elérte azt a szintet, hogy az 1992-ben megjelent Visio 1.0-tól kezdve a 2013-ban megjelenő Visio 2013-ig minden verzióban készült fájlt képes importálni. A LibreOffice 3.6-ban jelent meg először a CorelDRAW importszűrő. A 4.0-ban levő verzió azonban tartalmazza azokat a továbbfejlesztéseket is, amelyek lehetővé teszik, hogy az 1989-ben megjelent CorelDRAW 1.0-tól kezdve a legújabb CorelDRAW X6-ig minden verzióban készült fájlt képes importálni. Megjegyzendő, hogy erre a bravúrra az eredeti programok nem képesek, általában 1-2 verzióval korábbi fájlokat tudnak csak kezelni. A fájlformátumot pedig minden verzióban egy kicsit (néha nagyon) megváltoztatják. A LibreOffice 4.0-ban fog bemutatkozni az MS Publisher importszűrő, amely a Microsoft kiadványszerkesztőjével készített fájlokat tudja importálni. A libvisio, libcdr és libmspub programkönyvtárak önállóan is kiadásra kerülnek, lehetővé téve, hogy a más szabad szoftverek is felhasználják, például a Kalligra, a Scribus vagy az Inkscape.

Az előbbi bekezdésben említett importerek inkább az érdekes, mintsem a fontos kategóriába tartoznak, és szabadidős tevékenység (valamint a Google Summer of Code projekt) eredményeképpen jöttek létre. Azonban sok fizetett fejlesztő töltötte munkaidejét a felhasználók jelentős része által igényelt Microsoft Office formátumú (OpenXML; bináris DOC, XLS, PPT; RTF) fájlok kezelésének továbbfejlesztésével. Bár e fájlformátumok specifikációja elérhető, a leírás nem mindenhol pontos. Nem elég a leírásra támaszkodni, kísérletezni is kell. Olykor a Microsoft Office sem a leírást követi, nem az elvárt módon viselkedik, és a cél az volna, hogy a dokumentumok ugyanúgy jelenjenek meg LibreOffice-ban és Microsoft Office-ban. A LibreOffice 4.0-ban újdonság lesz a szövegtartományokhoz rendelt megjegyzések, valamint a „filctollal a lapra firkált" megjegyzések importálása DOCX-ből, és a natív RTF-képletek importálása RTF-ből.

Nemcsak hozzáadtak, el is vettek a támogatott fájlformátumok listájából. A LibreOffice 4.0 nem támogatja többé a StarOffice 1.0 – 5.0 bináris dokumentumait. Az ilyen formátumokba mentés képessége már korábban kikerült a programból, de mostantól megnyitni sem lehet ezeket a fájlokat. Talán mondhatjuk, hogy nem nagy veszteség, 2000 óta elavultnak számítanak. A funkció kivételét az indokolta, hogy amikor az OpenOffice.org 2.0-ban bevezették az OpenDocument Format támogatását, és emiatt olyan módosításokra kényszerültek, amelyek nem tették lehetővé a korábbi bináris dokumentumformátumok kezelését, az akkori fejlesztők az egész OpenOffice.org 1.1 forráskódját kicsit (de nem eléggé) lecsupaszítva egy binfilter nevű modulba szervezték ki, és ez biztosította a StarOffice 1.0 – 5.0 bináris formátumok exportálását és importálását. A LibreOffice fejlesztői úgy ítélték meg, hogy a szörnyen elavult kódot tartalmazó és egyre drágábban karbantartható binfilter modul kidobásának előnye felülmúlta a régi formátumok támogatásának elvesztésével járó hátrányokat. A binfilter modult törölték a forráskódból.

A LibreOffice 4.0-ban a Calc táblázatkezelő tetszőleges XML-fájlból képes adatokat importálni a munkalapra. Ez a funkció régóta hiányzott már. Az OpenDocument Format szabvány részét képező OpenFormula támogatása tovább javult. A LibreOffice 3.6-ban jelent meg a DATEDIF, IMTAN, IMSEC, IMCSC, IMCOT, IMSINH, IMCOSH, IMSECH és IMCSCH, ezeket a LibreOffice 4.0-ban a XOR függvény implementálása

követte. Egy új hozzájáruló teljesen váratlanul egy komplett opcióárazási függvénycsomaggal bővítette a Calc tudását (Black–Scholes modell). A LibreOffice 3.6-ban megjelent feltételes cellaformázás lehetősége a 4.0-ban további funkciókkal bővült, többféleképpen dekorálhatjuk ki a táblázatok celláit a megadott feltételektől függően, így jobban kiemelhetők a lényeges elemek.

A LibreOffice 4.0 újdonsága, hogy a program részévé vált a LibreLogo eszköztár. „A LibreLogo a magyar közoktatásban, sok helyen a felsőoktatásban is használt, zárt, licencdíjas, windowsos informatikai oktatóprogramok (Comenius Logo és Imagine Logo) szabad, és szabad operációs rendszereken is futó alternatívája. Mivel egyesíti a Logo és a Python programozási nyelv előnyeit, egyszerűbben oldhatunk meg vele informatikai verseny- és emelt szintű érettségi feladatokat is, mint a zárt Logo rendszerekkel. A LibreLogo lehetőségei azonban nem merülnek ki az oktatásban: interaktív (kézzel is átszerkeszthető) vektorgrafikus ábrákat készíthetünk vele nyomdai minőségben, kiadványszerkesztési céllal." – (idézet Németh László LibreLogo c. füzetéből, kiadta az FSF.hu Alapítvány 2012-ben[27]).

Az újdonságok felsorolása nem teljes, bár e cikkben többet felsorolhattam, mint az előadásban, ahol az idő szorított. A teljes lista talán nem is fog elkészülni, hiszen amint a statisztikáról szóló részben fejtegettem, nem egyszerű az alapadatokból kinyerni a teljes igazságot. Például sokszor nehéz eldönteni, hogy mi számít új funkciónak és mi hibajavításnak. Ez nézőpont kérdése is lehet. Ezek azok a dolgok, amiken fontosnak tartottam, amik felkeltették az érdeklődésemet, valamint úgy gondoltam, hogy kívülállóknak is érdekesek.

Az előadás és e cikk célja leginkább az érdeklődés és a lelkesedés felkeltése volt a LibreOffice projekt iránt. A LibreOffice fontos projekt, fontos hogy sikeres legyen. Akkor lehet csak sikeres, akkor tud fejlődni, ha egyre több ember kapcsolódik be a fejlesztésbe. A https://www.libreoffice.org/get-involved/ oldalon megtalálható minden információ, ami ehhez szükséges.

[27] 2013-as kiadás: http://www.numbertext.org/logo/logofuzet.pdf (szerk.)

HOGYAN KÉSZÍTSÜNK WRITER FUNKCIÓKAT?

Vajna Miklós

Kivonat

A nyolc lépés, amelyet minden új LibreOffice Writer funkció elkészítése során érdemes követnünk: dokumentummodell, UNO API, megjelenítés (layout), szűrők, felhasználói felület, tesztek, dokumentáció, specifikáció.

Tartalomjegyzék

1. Bevezetés

Nem csak a funkciók számítanak. Mielőtt belevetnénk magunkat a funkciók készítésének rejtelmeibe, ne felejtsük el, hogy nem ez az egyetlen módja a LibreOffice projekt segítésének. Segíthetjük a felhasználókat (levelezőlista, fórum, IRC), tevékenykedhetünk a minőség javításán (hibajelentések készítése, nem megerősített hibák reprodukálása), útmutatókat írhatunk, a design csoport pedig inkább csak a fejlesztéshez értő fejlesztőknek ad tanácsot jó felhasználói felületek kialakításához. Az infrastruktúrát üzemeltető önkéntesek nélkül a többiek nem tudnának tevékenykedni.

De még egy fejlesztőre váró kihívások közül is csak egy az új funkciók készítése: sok időt visz el a hibák javítása, tesztelés, a létező kód átrendezése, új önkéntesek segítése, dokumentálás. Ezek mind legalább annyira fontosak, mint az új funkciók fejlesztése. Ennek ellenére, ez a cikk a továbbiakban kizárólag ezzel a témával foglalkozik.

Mielőtt továbbhaladnánk, nézzünk két példát új funkcióra, amelyek a LibreOffice 4.0-ban fognak bemutatkozni:

- Korábban csak egy adott ponthoz fűzhettünk megjegyzéseket. Az új *hozzászólás szövegtartományhoz* funkció lehetővé teszi, hogy – egy kijelölés után – olyan megjegyzést készítsünk, mely eltérő kezdő- és végpozícióval rendelkezik.
- A *más élőfej/élőláb az első oldalon* funkció lehetővé teszi, hogy egyetlen oldalstílust használva ne csak a jobb/bal oldalon lehessen külön élőfej/élőláb, hanem az első oldalon is.

Mindkét funkciót régóta kérik a felhasználók – ezt tekinthetjük általános szabálynak is: egy, a 80-as évek óta fejlesztett irodai szoftvercsomagba érdemes meggondolni, milyen új funkcióra van szükség, csak akkor érdemes nekikezdeni valami újnak, ha arra valós igény van.

Mindezek után tekintsük át azt a nyolc lépést, amelyet minden új funkció elkészítése során érdemes követnünk:

1. dokumentummodell
2. UNO API
3. megjelenítés (layout)
4. szűrők
5. felhasználói felület
6. tesztek
7. dokumentáció
8. specifikáció

A cikk hátralevő részében ezeket a lépéseket részletezzük.

2. Dokumentummodell

Azt lehetne gondolni, hogy egy funkció fejlesztése során csak ezzel kell foglalkozni. A Writer klasszikus Modell-View-Controller paradigmában gondolkozik. Egy dokumentum egy `SwDoc` osztálynak feleltethető meg. Ezen belül az építőkockánk a bekezdések (`SwNode` osztály). A forráskódban ezek az `sw/source/core/` könyvtár alatt találhatóak.

Érdekesség, hogy nincs külön szöveges csomópont a szövegtartományoknak[28], he-

[28] Olyan szakasz egy bekezdésen belül, ahol a karakterek tulajdonságai is – pl. betűtípus – megegyeznek.

lyette egy `SwpHints` osztályba gyűjtve minden bekezdésen belül változó tulajdonsághoz egy `SwTxtAttr` tartozik, kezdeti pozíció, végpozíció és érték információkkal.

Ezen kívül még sok a kivétel, külön kollekcióban (nem bekezdésekhez csatolva) található meg például az oldalstílusok (`SwPageDesc` osztálya). Az „első oldali élőfej" funkcióhoz például ehhez az osztályhoz kellett az `aMaster` és `aLeft` tagok mellé egy `aFirst` tagot is felvenni.

Aki szeret példákon keresztül tanulni, annak érdemes az
`SW_DEBUG=1 ./soffice.bin`
opcióval indítania a Writert, ilyenkor a Shift-F12 a `nodes.xml` file-ba kiírja a jelenleg aktuális dokumentummodellt.

Másik lehetőség, ha UNO-n keresztül, makróból szeretnénk végigiterálni a bekezdéseken:

```
enum = ThisComponent.Text.createEnumeration
para = enum.NextElement
para = enum.NextElement
xray para
```

Így egy tetszőleges bekezdés tulajdonságait tekinthetjük meg.

3. UNO API

Makrókból nem férhetünk hozzá közvetlenül a Writer belső dokumentummodelljéhez, ehhez annak UNO API-ját kell használnunk. Ezért fontos, hogy az új funkciónkat elérhetővé tegyük ezen a publikus API-n keresztül is.

Ez egyébként is hasznos, mivel ha makróból már tudjuk írni/olvasni az új funkciót, akkor a felhasználói felület elkészítése előtt is már tudjuk azt tesztelni. Ezen kívül az UNO alapú (ld. később) szűrők csak az UNO API-n keresztül elérhető funkciókat tudják menteni, valamint a tesztesetek is az UNO API-t használják legtöbbször. A Writer UNO API kódját az `sw/source/core/unocore/` alatt találjuk.

Nézzünk példákat! Ha a környezetérzékeny térközöket szeretnénk bekapcsolni egy bekezdésre:

```
enum = ThisComponent.Text.createEnumeration
para = enum.NextElement
xray para.ParaContextMargin
para.ParaContextMargin = True
```

Ha szeretnénk, hogy az első oldalon más élőfej/élőláb legyen:

```
oDefault = ThisComponent.StyleFamilies.PageStyles.Default
oDefault.HeaderIsOn = True
oDefault.HeaderIsShared = False
oDefault.FirstIsShared = False
```

Ha megjegyzést szeretnénk csatolni egy szövegtartományhoz:

```
oTextField =
oDoc.createInstance("com.sun.star.text.TextField.Annotation")
oTextField.TextRange.String = "Tartalom"
oDoc.Text.insertTextContent(oCurs, oTextField, True)
```

4. Layout

A layout célja megjeleníteni azt, ami eddig csak a dokumentum modelljében volt elrejtve, azaz a „View" az MVC-ből. A Writer esetében ennek is saját dokumentummodellje van, aminek az építőkövei a keretek. Egy megnyitott dokumentum egy `SwRootFrm`-nek, egy oldal egy `SwPageFrm`-nek, majd ezen belül egy bekezdés egy `SwTxtFrm`-nek felel meg. Egy modell elem több layout elemnek is megfelelhet, gondoljunk csak arra

az esetre ha egy élőfej több oldalon is megjelenik, vagy egy bekezdés átfolyik a következő oldalra. Ehhez az `1:N` megfeleltetéshez a Writer az `SwClient` / `SwModify` mechanizmust használja: egy kliens csak egy kiszolgálóhoz lehet regisztrálva, viszont a kiszolgálóknak lehet több kliense, melyeken az `SwClientIter` osztállyal lehet iterálni.

Ha példát szeretnénk látni a layout felépítésére, a dokumentum modelljéhez hasonló módon megtehetjük, csak F12-re lesz szükségünk a Shift-F12 helyett.

5. Szűrők

Próbáljuk meg túlélni az újraindítást! Ehhez a szűrők úgy segítenek bennünket, hogy az exporterek egy `SwDoc` példányt mentenek egy streamre, az importerek pedig egy stream alapján felépítenek egy (eredetileg üres) `SwDoc`-ot.

Alapból minden szűrő „alien”, ami azt jelenti, hogy ismert, hogy információt vesztenek a konvertálás során. Egyetlen kivétel az ODF szűrő, amire emiatt „own”-ként hivatkoznak. Ennél a szűrőnél garantált a veszteségmentesség, ezért is kell kiterjeszteni *minden* új funkció implementálásakor.

A szűrők lehetnek UNO-alapúak (pl. DOCX import, RTF import), lehetnek beépítettek (belső sw API-t használják, pl. DOC import, DOC/DOCX/RTF export), végül lehetnek kevertek, mint amilyen az ODF szűrő (főleg UNO, de kis részben belső). Ezek forráskódjai az `sw/source/filter/`, `writerfilter/`, ill. az `xmloff/` alatt találhatóak.

6. Felhasználói felület

Eljutottunk oda, hogy a felhasználók számára is élvezhetővé tudjuk tenni a munkánkat. A felhasználói felület a színfalak mögött lehet régi vagy új.

A jelenlegi ablakok nagy része még a régi formátumot használja, tervezői vélhetőleg „ne használjunk egeret az UI elkészítéséhez!” felkiáltással tervezhették. Tipikusan egy `.src`, `.hrc` és egy `.cxx` file készült minden ablakhoz. Minden elem rögzített pozícióval / mérettel rendelkezett. Például:

```
CheckBox CB_SHARED_FIRST
{
HelpID ="svx:CheckBox:RID_SVXPAGE_HEADER:CB_SHARED_FIRST";
Pos = MAP_APPFONT ( 12 , 46 ) ;
Size = MAP_APPFONT ( 152 , 10 ) ;
Text [ en-US ] = "Same content on first page" ;
};
```

Számos probléma van ezzel a megoldással, például ha új elemet adunk az ablak közepéhez, az összes többi vezérlőt le kell mozgatni kézzel; vagy a gomb szélessége a leghosszabb fordítást kell figyelembe vegye stb.

Ezzel szemben sokkal előremutatóbb Caolán McNamara új Glade-alapú megoldása[29], bár ha csak egy-egy új vezérlőt adunk létező ablakokhoz, egyelőre ritkán találkozunk velük.

A felhasználói felület gyakran közös a LibreOffice egyes moduljai (Writer, Calc, Impress stb.) között, ezért nem írhatjuk alkalmazásfüggőre. Így a felhasználói felület és a dokumentummodell közötti interakcióra kitalált megoldás az `SfxItemSet` lett. Ha egy jelölőnégyzetet szeretnénk `SfxItemSet`-be tenni:

```
aSet.Put(SfxBoolItem(nWSharedFirst,
    aCntSharedFirstBox.IsChecked()));
```

[29] http://conference.libreoffice.org/talks/

Ez után kell az `SfxItemSet`-ből a dokumentummodellbe (jelen esetben `SwPageDesc`) rakni:

```
rPageDesc.ChgFirstShare(((const SfxBoolItem&)
    rHeaderSet.Get(SID_ATTR_PAGE_SHARED_FIRST)).GetValue());
```

Ellenkező irányban is hasonló megoldásra lesz szükségünk. Erre számos létező példa található: a közös kód az `svx/source/dialog/` és a `cui/` alatt, a Writer-specifikus kód az `sw/source/ui/` alatt.

7. Tesztesetek

Miért ismételnénk meg régi hibákat, ha választhatunk újak közül is? :-) A tesztesetek ebben segítenek minket.

A LibreOffice-ban unitcheckeket (*minden* részleges modul fordítás végén lefutnak), slowcheckeket (minden teljes fordítás végén futnak), valamint subsequentcheckeket (külön `make subsequentcheck`-re futnak csak) használunk. Új funkciók esetén legtöbbször egy új slowchecket készítünk: ez hozzáférést ad az UNO API-hoz, viszont minden fejlesztő gépén lefut, így széles tesztelést biztosít.

Tesztelés során legtöbbször a dokumentummodellt (UNO API) vagy a layoutot (XML kiírás + XPath kifejezés kiértékelése) vizsgáljuk. Ez utóbbira egy példa:

```
CPPUNIT_ASSERT_EQUAL(OUString("Első fejléc"),
    parseDump("/root/page[1]/header/txt/text()"));
```

Import / export tesztek esetén először minimális reprodukáló dokumentumot készítünk. A tesztelés során vagy importálunk, vagy egy import-export-import szekvenciát hajtunk végre, attól függően, hogy mit akarunk tesztelni. Ez azért jó, mert így a tesztelendő dokumentumot mindig fájlban tárolhatjuk (nem kódból kell felépíteni), valamint a tesztelést is egy létező `SwDoc` példányon tudjuk elvégezni, nem pedig egy valamilyen formátumban elmentett fájlt kell vizsgáljunk.

Az elkészült (importált) dokumentum modelljét az UNO API-val könnyen vizsgálhatjuk. Pl. ha valamilyen nem-ASCII karakter probléma volt, sokszor elég csak a dokumentum hosszát (karaktereinek) számát vizsgálni:

```
CPPUNIT_ASSERT_EQUAL(6, getLength());
```

A teszt megírása után a teszt futtatása következik. Ha sikeres, érdemes visszavonni a javítást vagy funkciót, és megbizonyosodni arról, hogy a teszt ezek után hibát jelez, vagy is a teszt jó. Ha ez stimmel, eldobhatjuk a visszavonást, és összevonhatjuk a két commitot, így később nem kell keresni, hogy melyik módosítást tesztelte a teszteset.

A Writer tesztjeit az `sw/qa/` alatt keressük.

8. Dokumentáció

A felhasználói dokumentáció a súgó, ami F1-re jelenik meg. Új tartalmat ehhez úgy érdemes hozzáadni, hogy a felhasználói felületen kikeressük a jövendőbeli szakasz szomszédos részeit, majd `git grep`-pel rákeresünk a létező tartalmakra a `helpcontent2/` alatt.

Fontos, hogy míg normál kód módosításakor elég a fordítás a változtatásunk teszteléséhez (a `linkoo`-nak köszönhetően), addig a súgó módosítása esetén a fordítás után egy telepítés is kell (`make dev-install`), mielőtt kipróbáljuk annak eredményét. Egy súgófájlon belül minden bekezdésnek egyedi azonosítót kell adni, ettől válik kezelhetővé a lokalizációs segédprogramok számára, és válik lefordíthatóvá más nyel-

vekre. Egyszerűen másoljunk le egy létező azonosítót, majd tetszőleges algoritmust választva tegyük egyedivé[30].

A fejlesztői dokumentációra mindössze annyi a követelmény, hogy minden új osztálynak legyen legalább egy egysoros doxygen leírása, hogy mit csinál – preferáltan minden publikus metódusra is, de ez már nem követelmény.

9. Specifikáció

Azért nem ezzel kezdünk egy új funkciót, mert mindent ODF-be mentünk, ami egy nyílt szabvány, és csak az kerül bele az ODF-be, amit már 3 független implementáció életképesnek mutat. Ennek megfelelően alapból a LibreOffice egy *ODF 1.2 Extended* formátumba ment, ami tartalmazza a saját kiegészítéseinket, majd ezeket javaslatként eljuttatjuk az OASIS-hoz.

Egy ilyen módosítási kérelemnek tartalmaznia kell, hogy mi a motiváció a változtatásra, a szabvány szövegéhez, valamint a RELAX NG sémához milyen változtatásokat szeretnénk eszközölni, végül részleteznie kell a változtatás várt hatását (visszafele kompatibilitás kezelése stb).

10. Elérhetőségek

A szerző ezúton is elnézést kér, hogy sok – a cikkben szereplő – témát csak érintőlegesen említett, csak a Writer belső működéséről vastag könyvet lehetne írni, a cél leginkább a figyelemfelkeltés volt. Az alábbi linkek további kérdések esetén remélhetőleg segítséget nyújtanak.

- LibreOffice honlap: http://libreoffice.org/
- A diák és ezen cikk elérhetősége: http://vmiklos.hu/odp/

[30] Egyszerű inkrementálás, ínyenceknek kvadratikus próba tökéletesen megfelel. :-)